Behavior Based Access Control for Distributed Healthcare Environment

Behavior Based Access Control for Distributed Healthcare Environment

By
MOHAMMAD HOSEIN YARMAND, B.Sc.

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree
M.Sc.

McMaster University

M.Sc. McMaster University, Hamilton, Ontario

TITLE: Behavior Based Access Control for Distributed Healthcare Environment

AUTHOR: MOHAMMAD HOSEIN YARMAND, B.Sc.

SUPERVISORS: Dr. Douglas Down and Dr. Kamran Sartipi

NUMBER OF PAGES: xi, 124

Abstract

Sensitivity of clinical data and strict rules regarding data sharing have caused privacy and security to be critical requirements for using patient profiles in distributed healthcare environments. The amalgamation of new information technology with traditional healthcare workflows for sharing patient profiles has made the whole system vulnerable to privacy and security breaches. Standardization organizations are developing specifications to satisfy the required privacy and security requirements. In this thesis we present a novel access control model based on a framework designed for data and service interoperability in the healthcare domain. The proposed model for customizable access control captures the dynamic behavior of the user and determines access rights accordingly.

The model is generic and flexible in the sense that an access control engine dynamically receives security effective factors from the subject user, and identifies the privilege level in accessing data using different specialized components within the engine. Standard data representation formats and ontologies are used to make the model compatible with different healthcare environments. The access control engine employs an approach to follow the user's behavior and navigates between engine components to provide the user's privilege to access a resource. A simulation environment is implemented to evaluate and test the proposed model.

Dedication

To my beloved family, Majid, Masoomeh, Shoeib, Fatemeh, Zahra and my dear aunt, Sedigheh.

Acknowledgements

I want to take this opportunity to thank those who helped me to complete this research. I would like to express my thanks to Dr. Douglas Down and Dr. Kamran Sartipi for their guidance and continuous support. We browsed a wide range of topics and had several discussions to narrow down my research to a concrete comprehensive work. During this research they taught me valuable academic skills that will certainly be beneficial to my future performance.

Also, special gratitude to Dr. Karim Keshavjee who aided me with his vast knowledge of the healthcare standards. We explored numerous documents to extract the data we required for my research. I also would like to thank Dr. Ann Mckibbon who helped me by providing effective information about data workflows in hospitals. I would also like to thank the examiner committee members, Dr. Norm Archer and Dr. Frantisek Franek for their help and time that they put on reviewing this thesis. I also wish to thank many friends and relatives who were always there for me during the hard moments I had.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The cost of healthcare in developed countries is rising rapidly due to the population's expectation for a higher quality of health service including: broad accessibility, customizability, cost efficiency, and most importantly reliability and security. Also, the requirement of integrating computer applications within the healthcare domain has caused health professionals to embrace quickly growing distributed information and communication technologies. The new proposals for national and international healthcare standardization meet most of these requirements by adopting new techniques such as service oriented architecture (SOA) which removes the need to consider the details of the particular web technology employed for each distributed system.

While solving the problem of interoperability among heterogeneous systems, SOA introduces many security and privacy issues which are the natural consequences of provision of customizability and availability of services. Regarding the confidentiality, integrity and availability requirements of patient data, a major concern is to avoid disclosure of these data to unqualified users and to protect the data from different attacks. Access by unauthorized users to patient data may result in misdiagnosis, delaying of treatment, or mistreatment. Other consequences may include denial of insurance coverage, loss of job opportunities, and financial problems [44].

Authentication and authorization methods at inter-/intra-organizational levels should be employed to provide the required security. In this context, several methods have been proposed in the literature which could potentially be a proper solution to the access control problem, however few of them consider the problem in distributed environments [17, 41, 54, 70]. Most access control methods only deal with static systems. However, dynamism and configurability are two requirements of models for distributed systems [41, 54, 70, 71]. It is essential that the definition and enforcement of access control policies take into account the heterogeneity and dynamicity of the environment [66].

1.1 Motivation and problem statement

Access control is the process of limiting access to the resources of a system only to authorized users, programs, processes, or other systems [44]. Large organizations contain several stakeholders, users, services, and resources. Scenarios and business processes need to access different resources and multiple users might be involved in completing a scenario. Therefore any mistakes in access control decisions potentially cause unauthorized disclosure of the data of multiple resources and breaching the privacy of many users. Several requirements must be satisfied to be able to make the access control decision. These requirements include defining the responsible roles, determining the tasks that can be performed by a certain team, and expressing various constraints that must be respected while performing a scenario.

Additional issues must be discussed when access control is considered in distributed environments. Due to the decentralization and distributed access requests in these environments, maintaining the integrity of resources becomes a challenging issue. In addition, each organization enforces its own set of privacy and security rules. These rules might conflict or overlap. An access control model which is used in these environments must respect and apply the rules of the involved organizations.

The access control model should be designed in a way that different organizations can interact with it. Every access control model requires to retrieve certain data from the environment of the target system (i.e. the system which uses that access control model). These data vary based on the functionalities that the access control model offers. For example if a model does not consider membership of users to teams, a list of teams need not be provided to the model. The access control model must provide an interface that is sufficiently generic and flexible so that all of the involved organizations can use it to enter their data.

When an access control model is applied on a System of Systems (SoS)[1], the integrity, flexibility, generality, and robustness of the model becomes critical. Due to the high complexity of these environments, any errors in the operation of the access control model are difficult to detect. The healthcare environment can be considered as an example of a SoS, as it is composed of various computer applications developed for different purposes, where each application might be supported by a different vendor (that makes the environment heterogenous). In this environment, the privacy rules for accessing patient data varies from province to province.

In order to apply a generic model to a particular domain, the specific requirements of that domain should be considered. In the healthcare domain, many international and national organizations have recently released standards to integrate different computer healthcare applications. They define the architecture, clinical data model, and transportation protocol for integration in distributed healthcare environments. Any access control model that aims to be widely used in this domain, should be compatible with the technical specifications of these standards. Other specific requirements of the healthcare domain (defined by the healthcare standards) such as delegation of access rights from patients to care givers or data access in emergency situations must be supported by the

[1]SoS are large-scale concurrent and distributed systems that are comprised of complex systems. SoS integration is a method to pursue in development, integration, interoperability, and optimization of systems to enhance performance in future scenarios [62].

access control model.

Given the above issues, the specific problem which is targeted in this thesis is stated as follows:

Propose an access control model for distributed systems that is interoperable with various data formats and security rules of the involved organizations, so that the model can be applied to different environments. In particular, the model must be able to satisfy the security requirements of the healthcare domain.

1.2 Proposed solution

A novel access control model for distributed environments is designed which is dynamic, system independent, and configurable based on the security factors that are captured from the environment. The model uses the interactions and data flows between the system entities (such as users, roles, and resources) to adjust the access control decisions. These interactions are recorded and analyzed in a particular way to extract the behavior of the entities.

The model is well-suited for healthcare environments. The healthcare standards obligations about the security area, architectural specifications, data modeling, and system integration are carefully studied to make the access control model compatible with these standards. The benefits of several access control models are embedded and modified in the proposed model to support the domain specific requirements of the healthcare environment. Using these benefits allows the model to make access control decisions based on constraints defined for roles, teams, contexts, and delegation rules.

The security factors that affect the access control decision are detected and modeled by a class diagram. The semantic interoperability that the model provides for the interaction of the organizations in the target distributed system, is achieved by using a common data model and ontologies and terminology systems in the healthcare domain (introduced in Chapter 2). An interface is designed to allow security administrators to interactively enter privacy and security rules, instead of hard coding the rules into the model. These two features (using a data model and providing an interface for entering rules) make the model independent of the data format of the security factors and the format of the privacy rules of the involved organizations.

A new concept called *user behavior* is introduced which follows special patterns on a sequence of recorded attributes for a user to visualize his activities. The results of analyzing these patterns are employed in making the access control decision. In order to extract and compose the *user behavior*, the fundamental definitions, models, and architecture of Context-aware Systems are employed. The contexts are modeled and the components required to extract the contexts are included in the proposed architecture. An architecture is offered for the access control model that supports the features introduced in this section. An example in the healthcare domain is described and used to explain the definitions and architecture components of the proposed model. The example considers access requests of a nurse to information about the patients hospitalized in a hospital department.

1.3 Contributions

The contributions of this thesis are as follows:

1. Proposing an access control model for distributed environments that can be used as an add-on to various configurations and can support the aggregated benefits offered by a number of existing access control models.

2. Applying the proposed model on the healthcare domain by following the healthcare standards and extracting healthcare security requirements.

3. Proposing a common data model for the entities involved in the access control decision making process and for classification of the effective security factors of the environment.

4. The concept of *user behavior* is introduced to model and employ the history of actions performed by the users of the system to configure the access control model based on users' requirements.

5. The proposed model is specified and implemented, using common technologies. The model is tested with simulated data.

1.4 Thesis overview

The remaining chapters of this thesis are organized as follows:

Chapter 2: introduces major international and national healthcare standards that express solutions for the integration of healthcare systems. This chapter describes the most recent security (particularly access control) specifications that are offered by these standards.

Chapter 3: the required background knowledge on the security and access control domain is explained. Several related works on available access control models and the models used in the healthcare domain are introduced. New security requirements are introduced that are not addressed by existing access control models and clarify the need for proposing a new model.

Chapter 4: provides the definitions and formal specifications of basic access control concepts and concepts that are used in the proposed access control model.

Chapter 5: extracts and explains access control requirements in distributed environments. This chapter expresses the security effective factors and offers a data model for classification of these factors. The architecture proposed for the access control model is introduced and the responsibility of each component of the architecture is explained.

The internal structures of all components are specified by indicating related algorithms and technologies. A discussion on how to employ the *user behavior* concept in making the access control decision is provided.

Chapter 6: discusses the requirements, mechanisms and outputs of the access control model from different perspectives and compares the features of the model with existing models.

Chapter 7: introduces the tool that is implemented for the model architecture. An explanation is provided on how to state different policies with one of the available policy specification languages. Access requests of a user are simulated and input into the tool to observe the operation of the blocks of the model architecture.

Chapter 8: gives concluding remarks, discusses future work, and presents other applications of the proposed model.

Appendix A: contains the algorithms used in the proposed access control model.

Chapter 2

Healthcare standards

Standards are generally required when excessive diversity creates inefficiencies. The healthcare environment has traditionally consisted of a set of loosely connected, organizationally independent units. Patients receive care across primary, secondary, and tertiary care settings, with little communication and coordination among the services. There are many pressures on healthcare information systems to reduce these inefficiencies such that the data collected for a primary purpose can be reused in a multitude of ways.

The healthcare industry has many organizations developing specifications and standards to support information exchange and system integration. These specifications are used to provide interoperability for a wide spectrum of healthcare applications. National and international organizations release standards to effectively integrate healthcare systems. Here the major standards which are used in this thesis are briefly introduced. The security (particularly access control) specification of each of the standards (if applicable) is stated.

2.1 HL7

Health Level Seven (HL7) [5] is an international community of healthcare experts and information scientists collaborating to create standards for the exchange, management and integration of electronic healthcare information. HL7 version 3 (also called HL7 v3), the HL7 messaging standard, offers a standard that is testable and provides the ability to certify vendors' conformance. HL7 v3 uses the Reference Information Model (RIM), an object model that is a representation of clinical data and identifies the life cycle of the events that a message will carry. HL7 v3 applies object-oriented development methodology on RIM and its extensions to create messages. A general description of the HL7 standard is beyond the scope of this thesis. However, we do provide here more details on two HL7 concepts: *refinement process* and *message structure* [4, 36].

2.1.1 Refinement process

The strategy for development of HL7 v3 messages and related information structures is based upon the consistent application of constraints to a pair of base specifications, i.e., HL7 RIM and HL7 Vocabulary Domains, and upon the extension of those specifications to create representations constrained to address specific health care requirements. Using the base specifications, the HL7 methodology establishes the rules for refining these base standards to arrive at the information structures that specify a Message Type. Figure 2.1 shows the refinement process specified in HL7 methodology, where the different parts are discussed below.

- *Domain Message Information Model* (D-MIM) is a subset of the RIM that includes a fully expanded set of class clones, attributes and relationships that are used to create messages for any particular domain.

- *Refined Message Information Model* (R-MIM) is used to express the information content for one or more messages within a domain. Each R-MIM is a subset of the D-MIM and contains only those classes, attributes and associations required to compose the set of messages.

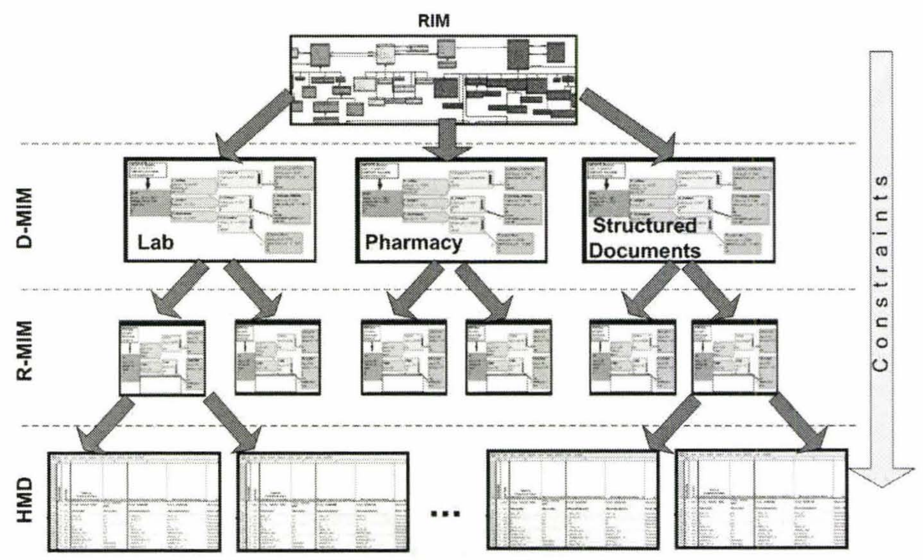


Figure 2.1: HL7 v3 information refinement process

- *Hierarchical Message Description* (HMD) is a tabular representation of the sequence of elements (i.e., classes, attributes and associations) represented in an R-MIM. Each HMD produces a single base message template from which the specific message types are drawn.

2.1.2 Message structure

Transactions consist of one or more messages to support both outbound and inbound communications (i.e. send/receive pairs). HL7 has suggested a structure for messages to support transporting interaction information and the actual payload. At the highest level, an HL7 v3 message is composed of two parts (see Figure 2.2):

- *HL7 Transmission Wrapper* includes the information needed by a sending application or message handling service to package and route HL7 v3 messages to the designated receiving applications or message handling services.

- *HL7 Transmission Content* is comprised of two parts:

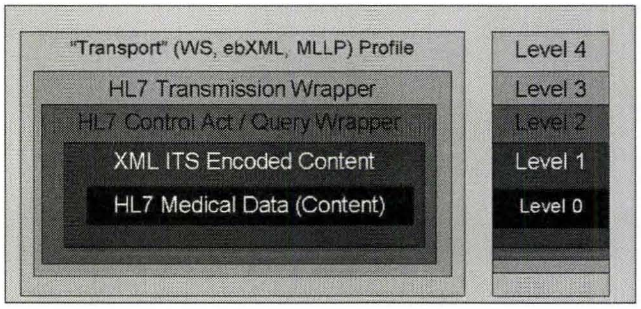

"Transport" (WS, ebXML, MLLP) Profile
HL7 Transmission Wrapper
HL7 Control Act / Query Wrapper
XML ITS Encoded Content
HL7 Medical Data (Content)
Level 4
Level 3
Level 2
Level 1
Level 0

Figure 2.2: HL7 v3 message structure [5]

- A "Trigger Event Control Act" contains the administrative information about the business event that initiated the sending of this message, who sent it and other associated business information.
- The "HL7 Domain Content" contains the domain specific content that is specified by the HL7 technical committee to satisfy a use case driven requirement for an HL7 messaging interaction. It includes the core data attributes for the message such as a prescription order or dispense event.

2.1.3 HL7 security

The HL7 security technical committee has specified an access control model based on user roles (called Role Based Access Control, explained in Section 3.1.2) to be applied in the healthcare domain. This committee suggested using scenario driven access control that is based on dividing scenarios to work profiles and tasks (further explained in Subsection 3.1.2). The access decision for running a scenario is dependent on the permissions that a user has about the work profiles that are associated with that scenario [38]. HL7 has released a list of healthcare scenarios that encompass security issues [37]. Also HL7 introduces a hierarchy of healthcare roles and defines their access privileges to different resources [39]. These specifications should be used together to determine the access rights of a role for completing an HL7 scenario.

2.2 Canada Health Infoway

Canada Health Infoway [1] is an organization that provides specifications for a standard, nationwide healthcare infostructure. The goal is to integrate information systems from different health providers and administrations (e.g., hospitals, laboratories, pharmacies, physicians, and government agencies) within each province, and then connect them to form a nationwide healthcare network with standard data formats, communication protocols, and a unique health history file for each patient; where the health information is accessible ubiquitously, using common services according to different access privileges for patients and providers.

Infoway has suggested a refinement of an Electronic Health Record Solution (EHRS) [47]. An **EHRS** is a combination of people, organizational entities, business processes, systems, technologies and standards that interact and exchange clinical data to provide effective healthcare. More technically speaking, an Electronic Health Record Infostructure (EHRi) should be defined to provide the technical framework for an EHRS. An **EHRi** is a collection of common and reusable components in the support of a diverse set of health information management applications. It consists of software solutions to support integration with the Electronic Health Record (EHR), data definitions for the EHR and messaging standards for integration and interoperability. An **EHR** provides each individual patient with a secure and private lifelong record of their health history. The record is available electronically to authorized health providers and the individual anywhere, anytime. Infoway's EHRi, called the Infoway infostructure, is shown in Figure 2.3.

2.2.1 Infoway security

Our work in this thesis specifies some of the requirements and services of the Health Information Access Layer (HIAL) of the Infoway infostructure. HIAL provides a single

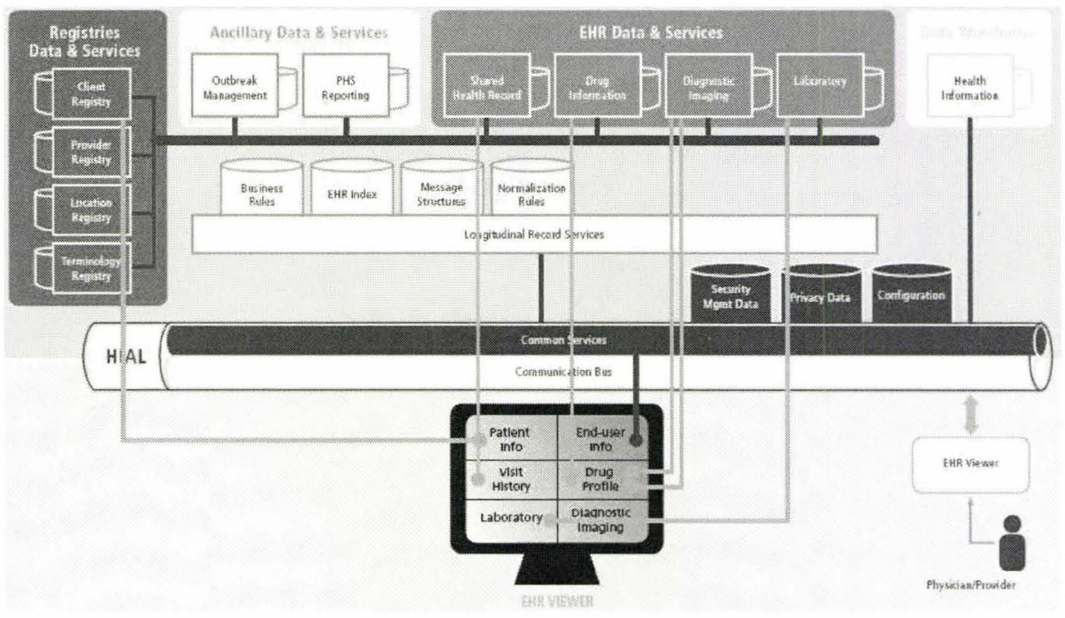


Figure 2.3: Canada Health Infoway Infostructure [47]

standardized way for Point of Service (PoS) applications to connect to the EHRi, regardless of how a particular jurisdiction has partitioned EHR information domains and services. HIAL provides standardized *common services* and *communication bus services* to sustain the interoperability of the different components within the infostructure, as well as to sustain interoperability and a high degree of abstraction between the EHR infostructure and the PoS applications. The *communication bus services* are divided into messaging (including transformation, routing, encrypt/decrypt, encode/decode, parser and serialization) and protocol (including applications protocol and network protocol). The *common services* of HIAL are shown in Figure 2.4. We focus on some of the services of the *privacy and security* part of this figure.

Different groups and projects are assigned to specify various components of the Infoway infostructure. The Privacy and Security Architecture (PSA) group is responsible for developing the security standards and maintaining information privacy. The PSA group has not yet suggested an architecture to serve security requirements but it has offered two useful documents: *EHR Privacy and Security Requirements* [45] which dis-

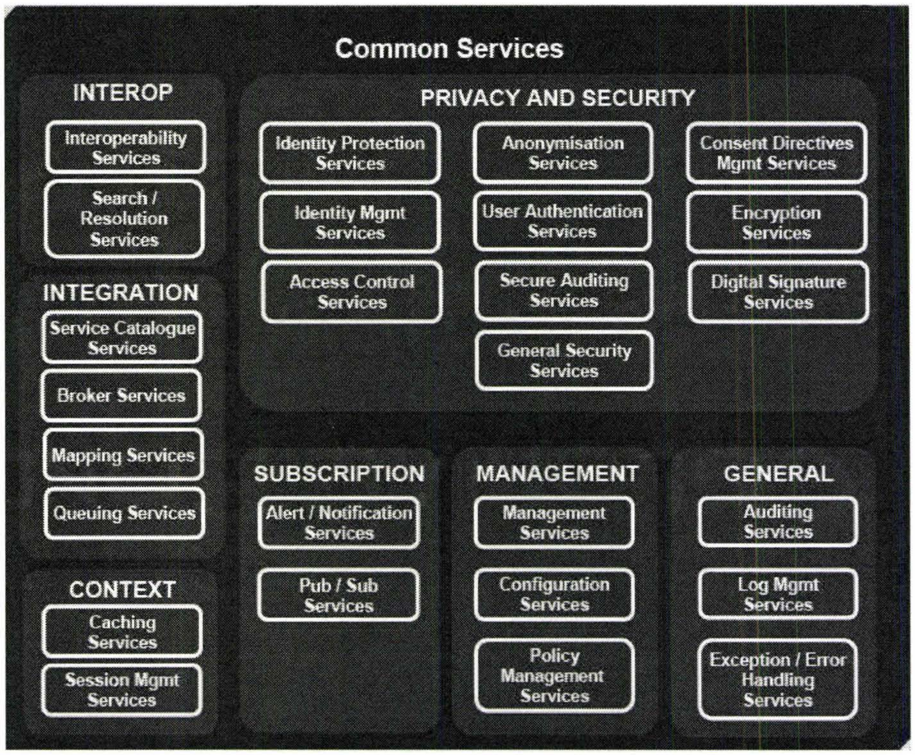


Figure 2.4: Health Information Access Layer (HIAL) common services [47]

cusses general security requirements in the healthcare domain and refers to data usage restrictions under privacy rules and *EHRi Privacy and Security Conceptual Architecture* [46] which expresses the specifications of the communication and common services shown in Figure 2.4.

The PSA group considers privacy requirements in the following areas: accountability for Personal Health Information (PHI); identifying purposes for collection, use and disclosure of PHI; consent; limiting collection of PHI; limiting use, disclosure and retention of PHI; accuracy of PHI; safeguards for the protection of PHI; openness about practices concerning the management of PHI; individual access to PHI; and challenging compliance. The security requirements considered by the PSA group are as follows: organizing information security; asset management; human resources security; physical and environmental security; communication and operational management; access control; information systems acquisition, development and maintenance; security incident management; compliance.

Focusing on access control specifications, the PSA group mentions several access control methodologies that must be provided as a part of a unified access control service. This service ensures the confidentiality and integrity of PHI. These methodologies are:

- role based access control, which relies upon the professional credentials and job titles of users established during registration to restrict users to those access privileges that are required to fulfil one or more well-defined roles. Professional roles, EHRi roles, and PoS system roles are differentiated as high level role categories.
- work group based access control, which relies upon the assignment of users to work groups (such as clinical teams) to determine which records they can access. Group-based access control allows users to be assigned to working groups such as a primary care clinic, the emergency department of a hospital, or a community-based health and social care team. Users can then rapidly be given access to all of the records of patients in the care of that team. Different formulations of work groups are geographic, organizational, and circle-of-care work groups.
- discretionary access control, which relies upon users with a legitimate relationship to a patient/person's EHR (e.g. a family physician) to grant access to other users who have no previously established relationship to that patient/person's EHR (e.g. a specialist).

Infoway has categorized consent based on the consent level and the actions that the consent directives management[2] must follow. The categories of these actions are: no

[2]The consent directives management service, one of the *common services* introduced in this subsection, is intended to help EHRi users and their organizations comply with the requirements in applicable legislation, as well as the requirements for the handling of PHI found in various privacy policies and in patients' specific consent directives. The service determines whether or not patients consent directives allow or restrict the use and/or disclosure of PHI. The service also allows EHRi users to manage a patients' specific consent directives, such as blocking or masking PHI from a certain care provider or disclosing PHI without consent for emergency treatment, as required or permitted by law.

consent, deemed consent, implied consent and express consent. The details are explained in the *EHRi Privacy and Security Conceptual Architecture* document [46].

2.3 HIPAA

The Health Insurance Portability and Accountability Act (HIPAA) [3] was enacted by the United States Congress and requires the establishment of national standards for electronic health care transactions and national identifiers for providers, health insurance plans, and employers. HIPAA provides a list of security and privacy suggestions and legal requirements. HIPAA explains non technical security and privacy aspects of medical systems with an emphasis on privacy.

These requirements include integrity, personal or entity authentication, transmission, access control, and risk analysis. The access control requirements suggested by HIPAA contain [35] unique user identification, emergency access procedure, automatic log-off, and encryption and decryption where the last two items are optional.

2.4 Clinical terminologies

Clinical terminologies are structured lists of terms which together with their definitions are designed to describe unambiguously the care and treatment of patients. Terms cover diseases, diagnoses, findings, operations, treatments, drugs, administrative items, etc. [8]. A clinical terminology system facilitates identifying and accessing information pertaining to the healthcare process and hence improves the provision of healthcare services by care providers. A clinical terminology system can allow a health care provider to identify patients based on certain coded information in their records, and thereby facilitate follow-up and treatment. Two major clinical terminologies are used in this thesis.

Systematized Nomenclature of Medicine – Clinical Terms (SNOMED CT) [59] is a comprehensive clinical terminology system that provides clinical content and expressivity for clinical documentation and reporting. SNOMED CT uses healthcare software applications that focus on collection of clinical data, linking to clinical knowledge bases, information retrieval, as well as data aggregation and exchange. The terminology is comprised of concepts, terms and relationships with the objective of precisely representing clinical information across the scope of healthcare. SNOMED covers a semantic network of over 300,000 medical concepts and their relationships. At the top level, there are 3 main hierarchies (finding, disease, and procedure) and 15 supporting hierarchies.

Logical Observation Identifiers, Names and Codes (LOINC) [7] was developed by the Regenstrief Institute, Indianapolis and the LOINC committee, to support the electronic movement of clinical data from laboratories to hospitals, physician's offices, and payers who use the data for clinical care and management purposes. The LOINC laboratory terms set provides a standard set of universal names and codes for identifying individual laboratory and clinical results. The LOINC database currently contains 41,000 observation terms where 31,000 of them are related to laboratory testing. LOINC is designed to be compatible with HL7.

2.5 IHE

Integrating the Healthcare Enterprise (IHE) [6] was formed in 1998 and initially focused on the domain of radiology. It has now become an initiative designed to promote standard-based methods of data integration in healthcare and encompasses industry and users. IHE offers a common framework to deliver the basic interoperability needed for local and regional health information networks. It also introduces a security framework for protecting the confidentiality, authenticity and integrity of patient care data. Another activity of IHE is Cross-Enterprise Document Sharing (XDS) of Medical Summaries to support the development of an interoperable patient summary within a context where much electronic clinical data is stored in proprietary formats.

The technical security specification of IHE [49] includes authorization (role management, user/role certificate management, assertion rights, delegation rights, validity time), node authentication (i.e. how to authenticate network connections), information access, information integrity (document update and maintenance policy, document digital signature policy, folder policy), ethics, audit trail, risk analysis, etc.

Chapter 3

Background and related work

The sensitivity of patient clinical data and strict rules regarding data sharing have made privacy and security a critical requirement for using patient profiles in distributed healthcare environments. From the wide range of security issues, we focus on the access control problem. In this chapter, literature is reviewed for general access control models, as well as the specific access control models for the healthcare domain.

In Section 3.1 the fundamental concepts of the access control domain are first explained. Then we review the access control literature. We mention those access control models which are adopted for use in the healthcare domain. The basic concepts for context-aware systems are described in Section 3.2. Several related work on context aware systems and context aware access control models are also mentioned in this section. Finally, in Section 3.3, several requirements are introduced that clarify the need for proposing a new access control model.

3.1 Access control models

In this section we first define some basic concepts in the access control domain. We then describe several access control models. Each model satisfies a particular access control requirement for distributed systems. Finally, some of the models used in the healthcare domain are presented. Most of the concepts that are introduced in this section are formally described in Section 4.2.

3.1.1 Preliminary definitions

The following definitions are used throughout this thesis.

Privacy. Privacy is the claim of individuals, groups or institutions to determine for themselves when, how, and to what extent information about them can be communicated to others [46]. In general, privacy rules describe an organization's data practices: what information they collect from individuals, for what purpose the information will be used, whether the organization provides access to the information, who are the recipients of any result generated from the information, how long the information will be retained, and who will be informed in the circumstances of dispute [43].

Policy. A policy describes the legal framework including rules, regulations and ethical aspects, the organizational and administrative framework, functionalities, claims and objectives, the principles (human users, devices, applications, components, objects) involved, agreements, rights, duties, and penalties defined as well as the technological solution implemented for collecting, recording, processing and communicating data in information systems [30].

Access control. Access control is the process of limiting access to the resources of a system only to authorized users, programs, processes, or other systems [44]. Access control includes identification of users during registration, their subsequent authentication during log in, and their authorization prior to being granted access to services and data. Access control is intended to prevent unauthorized access to information systems; ensure the protection of services; prevent unauthorized computer access; and detect unauthorized activities [46].

To gain a better understanding of the purpose of access control, it is worth reviewing the requirements of information systems. Information security risks can be broadly categorized into three types: confidentiality, integrity, and availability.

- **Confidentiality** refers to the need to keep information private. This category may include anything from state secrets to confidential documents, financial information, and security information such as passwords.
- **Integrity** refers to the concept of protecting information from being improperly altered or modified by unauthorized users.
- **Availability** refers to the notion that information is available for use when needed. For example, attacks that attempt to overload Web servers, are attacks on availability.

Access control is critical to preserving the confidentiality and integrity of information. The condition of confidentiality requires that only authorized users can read information, and the condition of integrity requires that only authorized users can alter information in authorized ways. Access control is less obviously central to preserving availability, but it clearly has an important role: An attacker who gains unauthorized access to a system is likely able to bring it down.

When considering any access control model one considers three abstraction forms for control: policies, models, and mechanisms. Policies are high-level requirements that specify how access is managed and who, under what circumstances, may access what information. At a high level, access control policies are enforced through a mechanism that matches a user's access request to a criteria, often defined by a simple table lookup, to grant or deny access. The mechanisms come in a wide variety of forms, each with distinct policy advantages and disadvantages (explained in Subsection 3.1.2). The models are written at a level of abstraction to accommodate a wide variety of implementation

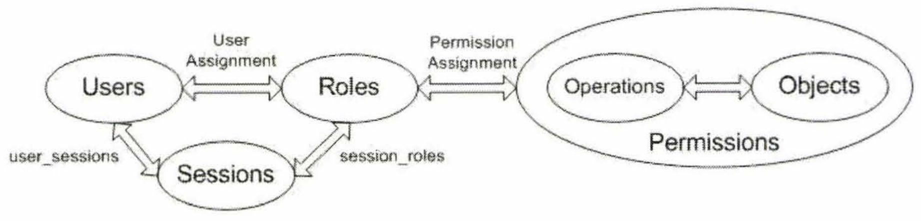


Figure 3.1: Role Based Access Control (RBAC) relations

choices and computing environments, while providing a conceptual framework for reasoning about the policies they support [29].

3.1.2 Generic access control models

In the following, several access control models are explained.

Role Based Access Control (RBAC). RBAC has been widely used (more than other models) in different systems and acts as a base for other models. According to research performed in 2007 that reviewed 351 articles [30], the most commonly used model is RBAC, being covered in 38 out of 52 selected articles. The preference for using RBAC as a starting point to build an access control model can be explained by the fact that this model allows easier administration and more flexibility in order to be adapted for heterogeneous environments. In RBAC, permissions (i.e. performing *operations* on *objects/resources*) are defined for *roles* instead of individual *users*. Once a user takes a specific role, the role privileges are transferred to the user. *Active roles* of each user are stored in an associated *session*. Figure 3.1 shows the relations between these concepts.

RBAC has many extensions such as Generalized RBAC, Generalized Spatio-Temporal RBAC [63] and Dynamically Authorized RBAC [54]. These models mainly consider additional environmental factors. This means that a user which normally has the privilege to access a resource, based on his role, must satisfy certain additional constraints so that the requested permission can be granted. These constraints include logical expressions (such as checking association between entities), and time and location restrictions.

Team Based Access Control (TBAC). This type of access should be considered when collaborative tasks exist that require accessing common resources. In this model permissions are assigned to different teams. When a user joins a team, he gains the privileges associated to that team in addition to his privileges, as long as he remains with the team. TBAC considers user team membership when deciding about user authorization [33].

Content Based Access Control (CBAC). CBAC makes access control decisions considering access restrictions that are defined for the resources and the content of the resources [34]. The concept of *restricted privilege* assigns a permission to a resource and checks for a constraint on the resource to allow accessing the resource regardless of the requester's identity. A *parameterized privilege* is a *restricted privilege* which accepts parameters in its constraint expression. A *role template* generalizes the concept of *role* by encapsulating and composing the parameters required by a *parameterized privilege.* The idea of Hippocratic databases[3] which might be dependent or independent of the users, is employed to embed the privacy into the data access layer. A simple example of these types of access rights is preventing the deletion operation on a particular resource when it contains certain data.

Attribute Based Access Control (ABAC). Whereas traditional access control models are based on the identity of the party requesting a resource, in open environments this approach is not effective because often the requester and the resource belong to different domains (and hence they might be modeled differently, e.g. their role hierarchy is not the same). In ABAC the access decision is based on attributes of the requester and of the resource. Basing authorization on attributes of the resource/service requester provides both the flexibility and scalability essential for large distributed open systems, where subjects are identified by their characteristics [25].

[3]The idea of a Hippocratic database was introduced by Agrawal et al. [13], and is founded on the premise that database systems should take responsibility for protecting the private data they manage. The authors describe ten principles governing the design of such a system. These principles are purpose specification, consent, limited collection, limited use, limited disclosure, limited retention, accuracy, safety, openness, and compliance.

Situation Aware Access Control (SAAC). SAAC is a model which uses RBAC to meet the dynamic, flexible and diverse security requirements of the users in distributed systems. *Situation* is defined as an expression on previous activities performed by a device over a period of time and/or the variation of a set of configurations relevant to the application software running on the device over a period of time. SAAC monitors situation changes through a situation-aware environment and enforces run time policies based on the extracted situation [70].

Scenario Based Access Control (SBAC). A scenario-driven process [56] is proposed with the goal of presenting a flexible systematic role engineering to avoid following ad-hoc approaches. Role engineering for RBAC is the process of defining roles, permissions, constraints and role-hierarchies. *Scenario* depicts system usage in the form of action and event sequences and serves as the base for this model. *Task* is defined as a collection of *scenarios* required to reach a particular goal that is performed by certain users of the system. *Work profile* is the complete set of all *tasks* that a specific kind of user is allowed to perform. *Work profiles* can be associated to the RBAC roles.

In distributed environments, *scenarios* and *tasks* require accessing resources from different organizations to complete their execution. When a user requests to perform a *task*, he should have sufficient privileges to access all resources of the organizations involved in the *scenarios* of that *task*. Otherwise he will not be able to perform one of the steps of the *task* and the whole *task* must be canceled [52].

Role-Based Delegation. *Delegation* is defined as the process whereby one active entity in the system authorizes another entity to act on behalf of the former by transferring a set of privileges. Decentralizing the administration of permission-to-user assignment is critical in distributed RBAC. The basic idea behind role-based delegation is that users themselves may delegate role authorities to other users to carry out some functions authorized to the former. The delegation rules define the conditions (e.g. preconditions that should be satisfied prior to delegation), the constraints (e.g. validity period of a delegation), and the quality (e.g. delegation depth) of delegation [72].

Context Aware Access Control (CAAC). This model is explained in Subsection 3.2.3 after presenting required introductory materials.

Context Sensitive Access Control (CSAC). This model is explained in Subsection 3.2.4.

3.1.3 Access control models for the healthcare domain

According to Ferreira et al. [30], who reviewed 59 articles on access control in the healthcare domain, 22 out of 40 selected articles used RBAC. A few of these access control models are described here.

Li et al. [55] extend RBAC for a laboratory information system. They define constraints on the *permission* relation. They extract the different roles and their associated job functions. The access privilege is narrowed down into accessing particular tables or data records within tables. Aspect Oriented Programming (AOP)[4] is used to implement this model. This allows a quite easy integration of the access control model into an existing system without changing the existing code.

Hung [43] provides an extended RBAC model that focuses on privacy in the healthcare domain to support confidentiality of patient data. He investigates the privacy requirements identified by HIPAA and by the International Organization for Standardization (ISO). He determines the major privacy concerns as 1) the acquisition, storage, and processing of data, 2) consent for processing and disclosure of data, 3) the rights of data subjects to access and modify their datasets. Schewartmann [64] proposes another version of extended RBAC called attributable RBAC. The idea is to attach constraints in the form of attribute-values (e.g. attending physician of patient x) to the permissions.

[4]AOP is based on the idea that computer systems are better programmed by separately specifying the various concerns (properties or areas of interest) of a system and some description of their relationships, and then relying on mechanisms in the underlying AOP environment to weave or compose them together into a coherent program. Concerns can range from high level notions like security and quality of service to low-level notions such as caching and buffering [28].

The whole model is formally defined. The functionality of the model is explained with a sample scenario of accessing a resource in the healthcare domain.

Hung et al. [44] discuss research issues in developing a privacy access control model for supporting mobile and ad hoc healthcare applications. Considering privacy rules for protecting Protected Health Informatics (i.e. limitation on collection, disclosure, use, and retention) and fundamental mobile properties (i.e. mobility, peer-to-peer, collocation, collaboration, transitory community), they provide a *policy enforcement and decision model* which contains a *policy decision point, a policy enforcement point, resources*, and *policies*. They use a policy specification language to specify and further implement their idea.

Blobel [21] proposes a generic access control model for electronic health record systems that deals with the policy description including policy agreements, authentication, certification and directory services. These elements form a privilege management infrastructure. Several models have been used to cover different requirements : the *domain* model, the *policy* model (i.e. an ontology for different types of policies), the *role* model, the *privilege management and access control* model (i.e. a class diagram for expressing relations between privilege management entities), and the *information distance* model. In order to obtain interoperability between the roles of different organizations, it has been suggested to map roles to the role hierarchy suggested by HL7 [20].

The CAAC models used in the healthcare domain [41, 51, 60] are explained in Subsection 3.2.3.

3.2 Context-aware systems

In this section a few fundamental concepts for context-aware systems are explained. We first define these concepts. Then we introduce a number of context classifications for different purposes. Various approaches to model contexts are expressed afterwards. In

Subsection 3.2.2, a general-purpose conceptual architecture is introduced that is required for sensing, extracting, and using the contexts. Finally, two access control models are explained which employ the concepts explained in this section.

Among different definitions for *context*, we accept Dey's [27] definition saying that context is: "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object including the user and applications themselves". Based on this definition of context, Dey defines *Context-aware Systems (CAS)* as follows: "A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task".

3.2.1 Context categorization

Each application identifies and categorizes different types of contexts based on the purpose of the application for using these contexts. Here we describe some of these classifications offered for different purposes. Wrona and Gomez [69] classify context information to *computing system* (including application, presentation, session, transport, network, data link, and physical), *user* (including task, social, personal, environmental), *environmental* (i.e. physical environment contexts such as lighting, and weather), and *temporal* (i.e. any context related to time). Kapsalis et al. [53] categorize contexts as *user*, *resource*, *environment*, and *history*. Historical context specifies previous events and situations, which constitutes an additional dimension of context information, including user and resource. The contexts used in different access control models [14, 33, 71] can be classified as represented in Table 3.1.

A context model is needed to define and store context data in a machine processable form. Some of the context modeling approaches are [65]: key-value models, markup schema models, graphical models, object oriented models, logic-based models, and ontology based models.

List of different contexts	
Context	Details
Location	from which access is requested where data/service server resides
Time	
Resource	Access pattern of the resource Sensitivity of the resource content Type of data/service the resource offers
Team	User context: membership of a user on a team Object context: the set of object instances that are required by a team to accomplish a task
User's intention	Service usage pattern
Communication network	Noise level / Link state Available bandwidth
Service provider server	Current load Availability Connectivity Available bandwidth

Table 3.1: Different contexts used in the access control domain

3.2.2 Context-aware System architecture

An architecture is required to detect and use contexts. Many CAS architectures have evolved recently. Most of them differ in functional range, location and naming of layers, the use of optional agents or other architectural concerns. However, a common architecture is identifiable. A layered conceptual architecture for CAS [15] is shown in Figure 3.2.

- *Sensors* layer is a collection of sensors, where sensor refers not only to hardware sensing, but also any data source which might provide usable context information (e.g. software applications and services).
- *Raw data retrieval* layer is responsible for the retrieval of raw context data. It uses appropriate drivers and Application Programming Interface (API) to make the low-level details of data retrieval transparent to components of the higher layers.
- *Preprocessing* layer is responsible for:
  - interpreting contextual information (converting the technical data returned by sensors to abstract data)
  - aggregation or composition of the information provided by some sensors (reason about the context of attendance in a conference based on the noise level and location contexts)
  - solving conflicting sensing when multiple sensors provide different values for an attribute
- *Storage* layer organizes data and offers them through an interface to the client (in synchronous and asynchronous methods).
- *Application* layer develops the actual reaction to different events and context-instances.

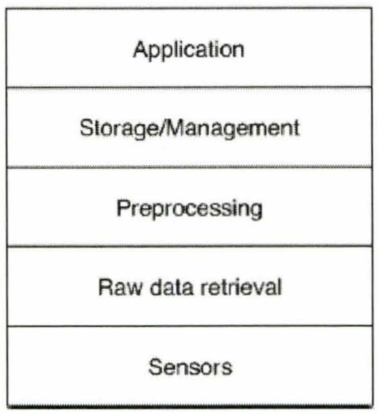


Figure 3.2: Layered conceptual framework for Context-aware Systems (CAS) [15]

3.2.3 Context-aware access control (CAAC)

CAAC authorizes users based on constraints defined over contexts. Different contexts are gathered from the environment and are evaluated against the constraints. If they do not satisfy these constraints, access is denied to the user. In the following we introduce several CAAC models.

Hu and Weaver [41] provide formal definitions of CAAC concepts and provide a context implementation hierarchy. This hierarchy is used to dynamically resolve authorization rules based on contextual constraints. They offer a layered architecture that uses contextual constraint checking in addition to traditional RBAC.

Al-Muhtadi et al. [14] offer a mechanism that integrates context-awareness with automated reasoning to perform access control in distributed environments. First-order logic predicate calculus and boolean algebra are used to express the contextual constraints. This allows the writing of various complex rules and evaluation of these rules in a manner similar to Prolog. Their architecture includes *context providers*, *context synthesizer*, *context history*, and *inference engine*. The *inference engine* uses application-specific access control policies, the credentials of the entity, and contextual information to decide whether an entity has access to a resource or not.

Zhange and Parashar [71] propose a dynamic model for CAAC. First they introduce a dynamic RBAC model by defining two separate state machines that represent the role and the permission hierarchies. The events that cause transitions between states of the state machines are defined based on the contexts. An example is limiting the access privileges of a user from *super user* to *normal user*, once it is detected that the communication link is not safe.

Bhatti et al. [17] aim to offer CAAC for Web services. They expand an extended RBAC model called XML-based Generalized Temporal Role Based Access Control (X-GTRBAC)[5] by adding contextual constraints. These constraints are specified by a grammar called X-Grammar. They formally define CAAC concepts and encode requests in the following tuple $< role, service\_name, context >$. They use the framework offered for X-GTRBAC as their architecture.

Jih et al. [60] offer CAAC for distributed healthcare environments. They explain a sample healthcare scenario and go through sensing and collecting contexts. The main elements of their proposed architecture are *access control manager, context manager, rule manager, rule engine,* and *data repository.* Jess [31] is used as their rule engine. The proposed context-aware rule engine is intended to run on resource-limited mobile devices. Therefore they evaluate the performance of their proposed model over existing mobile devices.

Jahnke et al. [51] provide a context-aware information service for healthcare environments. They mention that previous models had weaknesses in supporting knowledge sharing and context reasoning. They introduce an ontology-based context management system that allows a user to define contexts employing terms from the medical field. The core elements of their architecture are *context fact base, context pattern base,* and *context inference engine.* Their context ontology is composed of domain independent (e.g.

[5]X-GTRBAC [18] is an RBAC extension that provides a generalized mechanism to add temporal constraints on RBAC associations to meet dynamic access control requirements. Its XML-based framework makes it suitable to be configured to provide access control in Web services.

time and location) and domain dependent (follows the ontologies of HL7 RIM and HL7 Clinical Document Architecture (CDA)) parts and is represented in UML-like notation. They also discuss representation of contextual facts and context retrieval models.

Toninelli et al. [66] present a secure collaboration mechanism in distributed environments. They use CAS and semantic modeling technologies to provide a semantic CAAC framework. Semantic technologies are used for context/policy specification to allow high-level description and reasoning about contexts and policies. They offer a context ontology model and specify it using the Web Ontology Language (OWL). Description Logic (DL) and Logic Programming (LP) are combined to support the reasoning engine. They also propose a CAAC policy model that includes *policy specification, policy refinement*, and *policy evaluation* and use the above technologies (OWL, DL, and LP) for deployment.

3.2.4 Context-sensitive access control (CSAC)

In CSAC [42], context is both employed for user authorization and authentication (whereas in CAAC only the user authorization is based on context). In particular, subject authentication is based on verifying whether the claimed situational context is a valid context attribute of the subject. An example is authenticating a passenger on a moving train, by comparing his speed attribute value with another user which is already authenticated. Another possibility is to compare the proximity of the user to an authenticated user.

3.3 Motivating a new approach

As presented in this chapter, there are already several access control models proposed for the security and healthcare domains. A new access control model is necessary, only if the existing models can not satisfy the requirements of the target environment. Here we give some of the rules that we expect the access control model to be able to interpret:

- A user u_i has access to resource res_j only when he is a member of team tm_k between timestamps t_m and t_n.

- A user u_i (with active role r_i) can delegate his privileges to another user u_j (with active role r_j) for a maximum period of n hours, if r_i is a descendant of r_j in the role hierarchy.

- A user u_i has authorities beyond his normal privileges, if an unexpected situation is detected.

We could not find an existing access control model that satisfies all of these rules.

Another aspect of access control models is the administrative side. To the best of our knowledge, there has not been much effort toward improving access control models based on dynamic access requirements and characteristics of the target environment. In order to extract these requirements, it is necessary to visualize the activities of users of the system. For example, by monitoring denied access requests, it might be determined that the governing policy rule should be modified to allow a particular access. As another administrative concern, it is desirable to detect attempted attacks and suspicious behavior to prevent invalid disclosure of data. Also in the case that an unauthorized access occurred, the records stored for visualizing user activities must be analyzed to determine the cause of this occurrence.

With these issues in mind, we propose an access control model that satisfies these requirements. In the next chapter we provide our approach for visualizing the activities of users. Detailed explanations about the proposed model are presented in the following chapters.

Throughout this thesis, we use a running example to explain different concepts that are employed in the access control model. The example is used for clarifying definitions, architectures, specifications and implementations. Major entities of this example are: *Jane* and *Julie* as nurses; *Diabetes Department* as the department in which the nurses work; *Nero, Nancy* and *Nadia* as the patients hospitalized in the *Diabetes Department.* A typical scenario in our running example is that *Jane* requests to perform an operation on *Nero*'s profile. The activities of *Jane* are visualized and used in the process of making the access control decision. The appropriate privacy and security rules are fetched to evaluate the privileges that *Jane* requests.

Chapter 4

Access control model definitions

In this chapter a new access control model, well suited for the healthcare domain, is proposed for distributed environments. The proposed model is generic, system independent and configurable based on security factors. The model has interoperable functionality using common ontologies - providing ease of deployment in different healthcare institutions for the model. The model is bound to the configuration of a specific system by capturing characteristics of the environment in the input layer (explained in Section 5.2).

Ideally, system characteristics (data flow, workflow, and interaction between entities) can be observed to extract a security configuration that better serves the security requirements. Some of the system characteristics to consider are: interactions of instances of different roles, user distribution among roles, type of requested data or service, resource request frequency, delegation frequency between roles, denied access requests and data flow at inter and intra organizational levels. The access control policy can be automatically or manually configured based on these characteristics to satisfy dynamic user access requirements such as extending user access rights by adding new permissions, modifying delegation rules between roles, and increasing resource accessability.

In this chapter we define the basic concepts that are employed in the proposed model. Then we explain how to use these concepts to formally describe our model. In Section

4.1, the *user behavior* concept, employed to visualize user's activities, is informally defined. Different types of behaviors and their usage in making access control decisions, are described. In Section 4.2, we formally define the basic concepts of the security domain and also the *user behavior*. These definitions are further applied in the algorithms of the decision making engine of the model. An example is used to provide further insight.

4.1 Informal definitions

In this section, we define the concept of *user behavior* and describe how it can be applied to make an access control decision. Once we have a model for representing the behavior of a user, the data required by this model must be extracted for that user. When this data is organized as suggested by the behavior model, the behavior of that user is obtained. This behavior could be used for different applications. In this thesis we employ it for access control.

4.1.1 Action and behavior definitions

In order to capture the behavior of a user, we need to record a set of run time contexts of the user, whenever he interacts with the computer system. We call each of these interactions an *action* and represent it by a tuple called the *action tuple*. An *action* is any interaction which either requests a resource or changes the level of access privilege. The *action tuple* is composed of several *attributes*, as follows:

> *Action =<Person, Role, User Location, Server Location, Time of Day, Team, Delegation, Requested Profile Status, Service Invocation Type, Requested Data Type, Login/Logout Event>*

where *Person* is the user identification; *Role* is the user security role; *Server Location* is where the requested resource is located; *Team* refers to the team to which the user currently belongs; *Delegation* explains the access rights given or taken by the delegation rules or consents; *Requested Profile Status* refers to attributes of the *Resource Context* class explained in Section 5.2, for the requested profile; *Requested Data Type* refers to the clinical data type that the user has requested; *Service Invocation Type* is the type of service requested; *Login/Logout Event* records login or logout events. The value that each of these attributes takes, is called the *attribute value.*

A *Behavior* is defined as a sequence of actions that can be manifested in two forms:

- *Time-span behavior.* A record of a sequence of actions performed during a specified time, e.g., during the last five hours, a day, a month, etc. Each observation might include a portion of the attributes of the action tuple. For example, in one day different tasks performed by a person are recorded as action tuples and their collective effect is considered as behavior.

- *Snapshot behavior.* A record of particular attribute(s) of the "same action" in consecutive days to extract specific behavior over a long period of time.

Whenever an *attribute* of the *action tuple* of a user changes, a new tuple is recorded. Since we are modeling the privileges of the user, any changes in the set of user access rights should be monitored. A new tuple may be recorded even if the user has not requested access to a resource. For example when a user joins a team, his privileges change and therefore a new tuple should be recorded even if the user does not request access to a resource.

Both user-based and role-based behaviors should be captured. User-based behavior is needed to decide access rights based on an individual's behavior. Role-based behavior is needed to model the expected behavior of users who take a particular role. This can be either defined by the security administrator or can be automatically adjusted according to the expected behavior of users with the same role.

Having defined the *user behavior*, we define *common behavior* as follows: the behavior that a user is expected to follow based on the analysis of his behavior history. One of the approaches of this analysis is extracting the sequence of attribute values that an attribute takes most of the time, in the action tuples of a user (this approach is specified in Algorithm 4 of Chapter 5). *Common behavior* is also defined for both users and roles, depending on the data that is being analyzed. Available clinical guidelines can be used to evaluate the *common behavior* extracted for different roles.

4.1.2 Behavior based access control

The concept of user behavior can be used in different ways to make access control decisions. Here we introduce three different approaches.

Single action represents a single action tuple. Given a single action tuple we choose one of the action attributes as a key attribute and use it to constrain the domain of other attributes. If *Role* is the key attribute, the domain of other attributes like *Location, Service* and *Profile* would be limited based on *Role*. For example if the role is ophthalmologist, location can be limited to the ophthalmology department. In order to determine how attribute domains are filtered according to the *Role* value, general clinical guidelines or hospital policies defined for that specific role can be used. These guidelines suggest a list of valid attribute values associated to a role. The attribute values can also be extracted from the behavior history of each user.

If *Person* is the key attribute, the domain of other attributes would be limited based on that specific user. This makes our model dynamic and flexible in the sense that attribute domains are loaded for the specific user. In order to determine how the domains are filtered according to a specific user, the history of action tuples recorded for that user is analyzed to extract associated domain values. Although the access control decision in this case is made using a single tuple, it is important to note that the domain values extracted for attributes result from analyzing a set of tuples and are not independent of a user's general behavior.

Daily behavior consists of a sequence of action tuples recorded in one day for a given person. Some access control processes require more than a single tuple to be able to make an access control decision. Examples are: log in-out pattern; location proximity of consecutive requests i.e. considering issues such as the logical time required to reach different locations and make an access request; requested profile category and instance diversity, i.e. capturing if the user is requesting profiles of the same category and how many profiles he is working with; access request frequency; sequence of service invocation, i.e. matching a sequence of service invocations based on some initial service calls; policy rules explicitly defined over time such as access restrictions of a person on particular days; repetition of the same action attributes for similar cases. More specifically, the sequence of values that each attribute of the action tuple takes (in a day), is considered as a part of daily behavior.

Snapshot represents the historical aspect of our system. It considers the same attributes of actions of a person in consecutive days (called snapshot behavior).

In Chapter 5 we explain how to use these concepts for making the access control decision.

4.2 Formal definitions

In this section the basic concepts of the access control domain are defined, and the concepts introduced in the previous section are formally specified. Then an example is presented that uses the formal description of the model to process an access request. Finally, some algorithms used in the access control mechanism are provided based on the formal definitions.

4.2.1 Definitions

A number of primitive types are introduced to define the collection of *sets* required to provide a formal definition of the model. These primitive types are given in the following. We show how each primitive type is used in our running example. Table 4.1 lists the primitive types and relations that are introduced in this subsection.

- The set of users denoted by U where a user $u_i \in U$ is a person. For example $U = \{Jane, Nero, Nancy\}$.

- The set of roles denoted by R where a role $r_i \in R$ is a job function within the organization. Active roles in a session can be changed at the user's discretion. For example $R = \{user, patient, nurse\}$.

- The set of resources denoted by Res where a resource $res_i \in Res$ can be any system object which can be accessed, such as a file, printer, terminal, database record, etc. For example $Res = \{JaneAccount, NeroProfile, NancyProfile\}$.

- The set of sessions denoted by S where a session $s_i \in S$ is a temporary information about a user. Each session is assigned to only one user. For example $S = \{JaneSession, NeroSession, NancySession\}$.

- The set of teams denoted by T where a team $t_i \in T$ represents a group of users having specific roles with the objective of completing a specific activity in a particular context. For example $T = \{diabeticNursingTeam, researchTeam\}$.

- The set of delegations denoted by D where a delegation $d_i \in D$ is either a delegator or delegatee in a delegation relation. For example $D = \{delegator, delegatee\}$.

- The set of Logins/logouts denoted by L where a login-out $l_i \in L$ is either a login or a logout event in the system. For example $L = \{login, logout\}$.

- The set of data types denoted by DT where a data type $dt_i \in DT$ is a domain specific type of data (e.g. in the healthcare domain, different diseases or clinical observations) that exists in the system. For example
$DT = \{diabeticInfo, reminder, order\}$.

- The set of sensitivity denoted by Sen where a sensitivity $sen_i \in Sen$ is a member of a set that categorizes the sensitivity to different levels which are used for resources. For example $Sen = \{low, high\}$.

We now use the primitive types introduced above to express the entities and relations of the proposed access control model. These relations are finally employed to state an access request, an access control policy, and the access control method.

- Permission: $P \subseteq OP \times Res$, where $OP = \{read, append, delete, update\}$, is a relation which defines the operations that can be performed on resources. For example $P = \{p_1, p_2\}$ where $p_1 =< update, JaneAccount >$, $p_2 =< delete, JaneAccount >$.

- UserAssignment: $UA \subseteq U \times R$, is a relation which defines the roles a user can have. For example $UA = \{< Jane, user >, < Jane, nurse >, < Nero, patient >, < Nancy, patient >\}$.

- PermissionAssignment: $PA \subseteq P \times R$, is a relation which defines the permissions a role can have. For example $PA = \{< Jane, p_1 >, < Jane, p_2 >\}$.

- SessionUser: $S \mapsto U$, is a function mapping each session to a single user. The user is constant for a session lifetime.

- SessionRole: $S \mapsto 2^R$, is a function mapping each session to the set of active roles. The set of active permissions for a session can be inferred by using a combination of the UA and PA relations.

- RoleTeamAssignment: $RT \subseteq R \times T$, is a relation which defines the roles that can participate in a team. For example $RT = \{rt_1\}$ where $rt_1 =< nurse, diabeticNursingTeam >$.

- UserTeamAssignment: $UT \subseteq U \times RT$, is a relation which defines the set of users that participate in a team and their role in the team. For example $UT = \{< Jane, rt_1 >\}$.

- Context C, is a set of context parameters. In its simplest form the context parameters are *Location* and *Time* (the format of time is year.month.day-hour:minute). For example $c_1 =< diabeticNursingStation, 08.03.28 - 09:05 >$.

- Action: $A \subseteq U \times R \times C \times T \times D \times RC \times Res \times OP \times DT \times L$, is the action tuple defined in Section 4.1. For example $a_1 =< Jane, nurse, c_1, diabeticNursingTeam, nil, rc_1, NancyProfile, append, \{diabeticInfo\}, nil >$

- AccessPattern: $AP \subseteq 2^C$, is a subset of location-time pairs (is employed to determines the location and time stamp of the users who access a resource). For example $ap_1 = \{c_1\}$.

- ResourceContext: $RC \subseteq Res \times AP \times Sen \times 2^{DT} \times 2^U \times DT$, is a collection of attributes and contexts recorded for resources. For example $rc_1 =< JaneAccount, ap_1, low, \{reminder, order\}, Jane, reminder >$.

- Behavior: $B : U \mapsto 2^A$, is a function mapping a user to a sequence of actions that composes the behavior of the given user u. For example $b_1(Jane) = [a_1, a_2, a_3]$.

- $ActionCondition := (att)(op)(value)$ where *att* refers to an attribute of the action tuple, *op* is a logical operator in the set $\{>, \geq, <, \leq, \neq, =\}$, and the type of *value* is the same as *att*.

- ActionConstraint: *AC*, is an expression consisting of a 'and' and/or 'or' of *ActionCondition* expressions. For example $ac_1 =$ (team $= diabeticNursingTeam$) $\wedge$ (data type $\neq confidential$).

- AttributeSequence: $AttSeq : U \times ATT \mapsto 2^{att_{val}}$, is a function returning the sequence of attribute values (att_{val}) of the attribute $att \in ATT$ in the behavior records of user u, where ATT is the set of action tuple attributes. For example $attSeq_1(Jane, team) = [< diabeticNursingTeam, researchTeam >]$. The graph representation of the *AttSeq* is as follows:

  Graph $G = (V, E)$ is a directed graph with vertices V and edges E: each vertex v represents an action tuple attribute and is labeled with the value of the $Context.time$ attribute of that action tuple. There is a directed edge from v_i to v_j iff $v_i.label \leq v_j.label$ ($v_i.label$ refers to the label of vertex i). In order to extract the sequence of attribute values of an attribute, the graph G should be traversed in the following manner: first find those vertices which represent an attribute with the same type as the given attribute (att). Starting from the vertex with the smallest label among found vertices, then follow the outgoing edge to reach the next vertex, until there are no more edges to follow. The sequence that the *AttSeq* function must return is equivalent to the order of visiting the vertices of graph G, according to the described approach, and returning the corresponding values of the vertices.

- BehaviorConstraint: *BC*, is defined based on *AttributeSequence*. It describes the relations which should hold between the attribute values of action tuples in a given behavior history. For example $bc_1 = attSeq_1$ should start with $diabeticNursingTeam$. BC requires computing the distance between the extracted sequences for *daily behavior* and *common behavior*. This distance is computed for each attribute of the action tuple (we call it $dist_i$). The constraint is satisfied if:

  $\sum_{i=1}^{n} dist_i \leq threshold$ where $n =$ size of common behavior sequence.

- LogicalConstraint: LC, is a logical expression that can not be expressed with $ActionConstraint$ and $BehaviorConstraint$. Constraints that use resource contexts are defined here. For example $lc_1 = isMemberOfDiabetesDepartment$.

- $Constraint := LC \wedge AC \wedge BC$. For example $constraint_1 = lc_1 \wedge ac_1 \wedge bc_1$.

- AccessControlPolicy: $ACP \subseteq Subject \times P \times Constraint$, where $Subject$ is a user or a role. $AccessControlPolicy$ is a relation which defines the constraint that should be satisfied so that a subject can gain a permission. For example $acp_1 =< Jane, p_1, constraint_1 >$.

- AccessRequest: $AR \subseteq U \times activePerm \times B(u) \times A$, where $activePerm$ is the set of active permissions of a user (that is gained by applying $SessionUser(s)$, UA, and PA). This represents the request of a user to perform an operation on a resource. For example $ar_1 =< Jane, nurse, b_1(Jane), a_1 >$.

- **Access control mechanism**: an access request $ar =< u, p, b(u), a >$ is granted if an access control policy $acp =< s, p\prime, l >$ exists, such that $u \in s$, $p = p\prime$, and l evaluates to true under $b(u)$ (i.e. when the parameters of $ActionConstraint$ and $BehaviorConstraint$ are replaced by the values indicated by $b(u)$, the resulting boolean expression is true).

4.2.2 Examples

In this subsection, we use the formal definitions to explain an instance of processing an access request. Through this example, we show sample elements of the sets, relations and functions introduced in the previous subsection. Then we employ them to model user behavior and make the access control decision.
Suppose the primitive types are initialized as follows:

Primitive types and Relations	
Notation	Name
U	The set of users
R	The set of roles
Res	The set of resources
S	The set of sessions
T	The set of teams
D	The set of delegations
L	The set of login/logouts
DT	The set of domain specific data types
Sen	The set of sensitivities
P	The Permission relation
UA	User-Role assignment
PA	Permission-Role assignment
$SessionUser$	A function mapping a session to a user
$SessionRole$	A function mapping a session to roles
RT	Role-Team assignment
UT	User-Team assignment
C	The set of contexts
A	The action tuple relation
AP	The access pattern relation consisted of time-location pairs
RC	Resource context relation
B	A function mapping a user to a sequence of actions
AC	Action constraint expression
$AttSeq$	A function mapping a user and an action tuple attribute to a sequence of attribute values
BC	Behavior constraint expression
LC	Logical constraint expression
$Constraint$	$AC \wedge BC \wedge LC$
ACP	Access control policy relation
AR	Access request relation

Table 4.1: List of primitive types and relations

Users: $U = \{Jane, Nero, Nancy\}$

Roles: $R = \{user, patient, nurse\}$

Resources: $Res = \{JaneAccount, NeroProfile, NancyProfile\}$

Teams: $T = \{diabeticNursingTeam, researchTeam\}$

Data types: $DT = \{diabeticInfo, reminder, order\}$

Sensitivity: $Sen = \{low, high\}$

The following instances of relations are defined based on the above primitive types:

Permissions: $P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$ where $p_1 =< update, JaneAccount >$, $p_2 =< delete, JaneAccount >$, $p_3 =< append, NeroProfile >$, $p_4 =< append, NancyProfile >$, $p_5 =< read, NeroProfile >$, $p_6 =< read, NancyProfile >$

User-Role assignment: $UA = \{< Jane, user >, < Jane, nurse >, < Nero, patient >, < Nancy, patient >\}$

Role-Permission assignment: $PA = \{< Jane, p_1 >, < Jane, p_2 >, < Jane, p_3 >, < Jane, p_4 >, < Nero, p_5 >, < Nancy, p_6 >\}$

Role-Team assignment: $RT = \{rt_1\}$ where $rt_1 =< nurse, diabeticNursingTeam >$

User-Role-Team assignment: $UT = \{< Jane, rt_1 >\}$

"Suppose the user *Jane* has requested to update her account at 9:05. She then joins the diabetes nursing team and reviews *Nero*'s profile at 10:00, and appends some information to *Nancy*'s profile at 11:00". These access requests are respectively assigned to the action tuples a_1, a_2, and a_3:

Context1: $c_1 =< diabeticNursingStation, 08.03.28 - 09 : 05 >$

Access pattern1: $ap_1 = \{c_1\}$

Resource context1: $rc_1 =< JaneAccount, ap_1, low, \{reminder, order\}, Jane, reminder >$

Action1: $a_1 =< Jane, user, c_1, nil, nil, rc_1, JaneAccount, update, \{reminder, order\}, nil >$

Context2: $c_2 =< diabeticNursingStation, 08.03.28 - 10 : 00 >$

Access pattern2: $ap_2 = \{c_2\}$

Resource context2: $rc_2 =< NeroProfile, ap_2, low, \{diabeticInfo\}, \{Nero, Jane\}, diabeticInfo >$

Action2: $a_2 =< Jane, nurse, c_2, diabeticNursingTeam, nil, rc_2, NeroProfile, read, \{diabeticInfo\}, nil >$

Context3: $c_3 =< diabeticNursingStation, 08.03.28 - 11:00 >$

Access pattern3: $ap_3 = \{c_3\}$

Resource context3: $rc_3 =< NancyProfile, ap_3, high, \{diabeticInfo\}, \{Nancy, Jane\}, diabeticInfo >$

Action3: $a_3 =< Jane, nurse, c_3, diabeticNursingTeam, nil, rc_3, NancyProfile, append,$ $\{diabeticInfo\}, nil >$

User behavior: $b_1(Jane) = [a_1, a_2, a_3]$

The action tuple a_3 is recorded when the following access request (i.e. *Jane* requests to append some information to *Nancy*'s profile) is generated:

Access request: $ar_1 =< Jane, nurse, b_1(Jane), a_3 >$

Suppose the following access control rule exists in the rule repository:

Access control policy: $acp_1 =< Jane, p_4, constraint_1 >$

Constraint: $constraint_1 = lc_1 \cap ac_1 \cap bc_1$

Logical constraint: $lc_1 = isMemberOfDiabetesDepartment$

Action constraint: $ac_1 = (\text{team} = diabeticNursingTeam) \wedge$ (data type $\neq order$)

Attribute sequence: $attSeq_1(Jane, team) = [< diabeticNursingTeam, researchTeam >]$

Behavior constraint: $bc_1 = attSeq_1$ should start with $diabeticNursingTeam$

Following the definition of *accessRequest* as $< u, p, b(u), a >$ and the general definition of *accessControlPolicy* as $< s, p\prime, l >$, the access control mechanism is as follows: given the access request ar_1, it can be observed that:

equivalent to $u \in s$: $Jane \in \{Jane\}$

equivalent to $p = p\prime$: $< Jane, p_4 >=< Jane, < append, NancyProfile >>$

and for satisfying l the following statements from $constraint_1$ should be considered:

for ac_1, (team = *diabeticNursingTeam*) $\wedge$ (data type = *diabeticInfo*), evaluates to true for ac_1

for bc_1, $attSeq_1(Jane, team)$ starts with *diabeticNursingTeam* which holds for bc_1

for lc_1, *Jane* is a member of Diabetes Department which makes lc_1 true

Therefore $constraint_1$ is satisfied.

This means that access is granted for the requested action and *Jane* is allowed to access *Nancy*'s profile.

4.2.3 Formal definition application

In order to demonstrate the usage of the definitions offered in the previous subsection, here we specify two algorithms for the proposed access control model. These algorithms are further used in the access control decision making process, as explained in Section 5.3.

Algorithm 1 specifies RBAC related checks. The algorithm accepts a 'user' and the 'requested permission' as input and checks for active permissions in the user's session. As output the algorithm returns true, if the requested permission can be found in the set of active permissions for the user. Otherwise, it returns false. In this algorithm, $SessionUser^{-1}$ is the inverse of $SessionUser$ function considering that $SessionUser$ is a one-to-one function.

The access privileges that a user might gain by joining a team are considered in Algorithm 2. This algorithm receives the 'user identity', 'current role', 'current team' and 'requested permission' as input. The algorithm returns the access right based on the membership of the requested permission to the set of permissions allowed for the given role to perform in the given team.

Chapter 5

Access control model architecture

In this chapter an architecture is introduced that supports the features of the proposed access control model. A common data model is created to provide interoperability for collecting the information that the architecture requires. The architecture specifies how to apply the definitions offered in Chapter 4.

The remainder of this chapter is structured as follows. Section 5.1 introduces a number of security requirements in distributed systems. The security effective factors that we consider in our model are introduced and categorized in Section 5.2. A common data model for representing these factors is also offered in this section. Section 5.3 explains the architecture of the proposed model in detail. Each component of the architecture is specified separately. Finally, a usage scenario is offered in Section 5.4 to observe how the architecture processes a request.

5.1 Distributed systems security requirements

As a first step for developing an adaptable access control model, it is essential to identify and analyze the requirements. Several researchers have discussed these requirements [40, 55, 61, 71]. Here we provide a summary.

Distributed access requests. A resource is accessed and possibly modified by different users of a single or multiple organizations. Therefore a process is required to maintain the integrity of the resources over all interactions. This requires integrity in authentication of users and integrity in communicating data. In the healthcare domain different tasks such as diagnosis, therapy prescription, therapy validation, and drug administration can modify patient profiles from different Points of Services of a distributed system.

Organization specific privacy rules. In a distributed environment, each organizations has its own privacy rules, which might result in conflicting policies over different situations. Resolving such conflicts and at the same time satisfying the different privacy rules, can not be offered by traditional models and requires an intelligent access control model. In the healthcare domain, different provinces have their own privacy rules for patient data. For example in Canada, Quebec uses express consent (i.e. directly given either orally or in writing) while Ontario employs implied consent.

Context awareness. Access control decisions depend on different environmental parameters such as constraints between the subjects, objects, permissions, locations, and time. These context-related data are necessary to perform integrity checks at the point of service. For example, a medical student can not view a patient's profile unless he is co-located with that patient's attending physician. These situations require a context aware infrastructure to enforce the necessary policies. The context of a user determines the set of rules that should be used for making the access control decision.

Sequence control. A portion of access requests obligates dynamic and run time checking of performed actions. They include detection of emergency and special situations and scenarios, following the history of invoked services, etc. For example, a care giver cannot invoke a service to retrieve information about a patient unless he has already invoked the service for signing the confidentiality agreement. A special case of making an access control decision using sequence information is granting permission to invoke a service at most a fixed number of times in a given period. An example is preventing a surgeon from performing more than a certain number of surgeries per day.

Generality. Since organizations involved in an intra organizational workflow are subject to change, removal or modification, the access control model should be general and flexible enough to be compatible with different access control requirements. Also the workflow should not depend on any specific mechanism to provide Separation of Duty (SoD)[6] in the application layer.

Decentralization. In order to cover the requirements of all involved organizations, a decentralized and distributed access control method should be used. A mechanism is required to locate, retrieve and authenticate the policy components from different parties.

Flexibility of policies. Policies should not be hard-coded and a security administrator should be able to modify system policy rules via an API. This API should permit new policies to be dynamically and easily specified on demand as new situations occur as well as allowing existing policies to be modified to adapt to changing conditions. The policy engine must support real time policies. For example, let us consider the case of a meeting that continues beyond its originally scheduled end time. It is essential to ensure that meeting participants can continue to access each other's resources as long as the meeting is actually taking place.

Temporal roles. Some permissions might temporarily be assigned to users in special conditions and expire on satisfaction of another condition. For example a visiting doctor can treat a patient who is not normally his/her patient if there is a pressing reason and the patient agrees. This access right should not be permanent, however.

Auditing. Access requests, granted accesses and denied accesses should be recorded centrally. These records would be useful for electronic fraud detection and also for identifying the cause of an invalid decision or an action taken by the system.

Having these requirements in mind, we offer an access control model that satisfies most of these requirements, with particular focus on the dynamic aspect.

[6]Separation of Duty is the concept of having more than one person required to complete a task. More specifically it is defined as disseminating the tasks and associated privileges for a specific business process among multiple users [22].

5.2 Security effective factors

In order to keep the model as general as possible, different security factors should be both identified and captured in different environments. They are used as parameters for the privacy and security rules. These factors are modeled in the class diagram shown in Figure 5.1. This class diagram shows the interaction of security effective factors involved in making access control decisions. The elements of the proposed model are tightly and logically related to the data flow in the system. A clear and accurate representation of security factors and their inter-relationships are necessary for effective operation of other blocks of the model. The class diagram is specific to the healthcare domain, however it can be generalized to other domains by using the appropriate standards.

In order to establish interoperability and reusability, the relations between these input factors and standard clinical data are defined, i.e., our class diagram is connected to the standard RIM classes. A few classes of HL7 RIM have been used in this class diagram. The *service type* class (top right) represents a list of services that a user invokes; this list is mapped to Infoway storyboards and transactions of different domains covering standard healthcare scenarios [48]. The *data type* class (top right) expresses the type of clinical data using the higher levels of standard clinical terminology hierarchies such as SNOMED and LOINC [24, 58].

There are four categories of classes in the proposed class diagram: i) HL7 classes which are labeled by *(RIM)* and located at the top of the class diagram; ii) context hierarchy classes at the bottom of the class diagram that represent different contexts; iii) core security classes in the middle of the diagram, and; iv) enumeration classes on the right side. We extend the policy classification offered by the Ponder project [26] to represent different policies. The remainder of this section explains the major classes.

The *role(RIM)* class, defined in HL7, represents the general role of a person, such as physician. However the *role* class in our model refers to the security role which is a subset of *role(RIM)* and therefore the inheritance relation holds between the two classes.

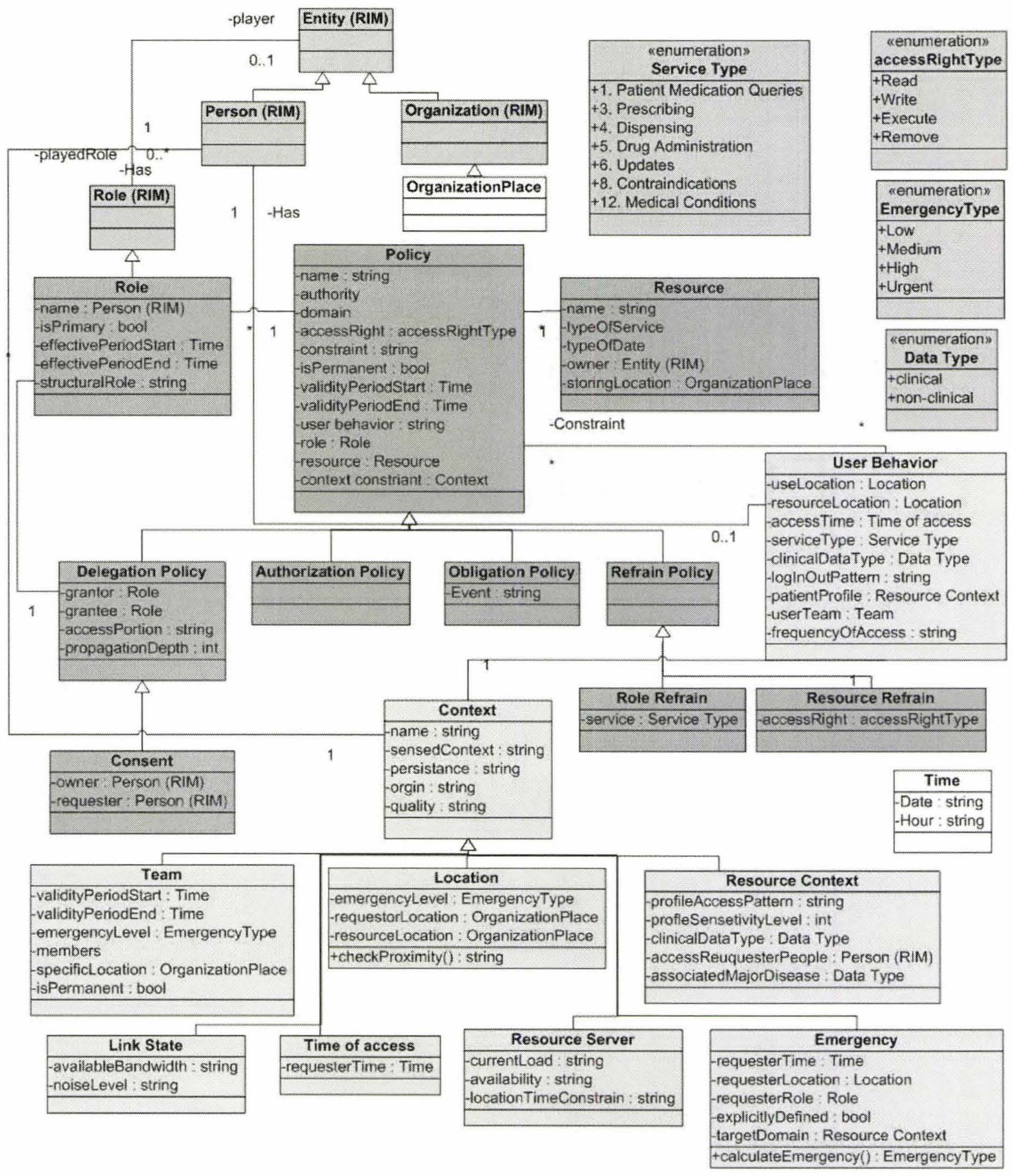


Figure 5.1: The class diagram of the security effective factors

This class can represent either a functional or structural role. *The organizationPlace* inherited from *organization(RIM)* represents the internal locations of a hospital such as emergency room, operating room or nursing station. For the *resource* class, the type of data or services it offers, the organization or person who owns the resource and the resource location are stored.

The *policy* class is an association class between the *Role* and *Resource* classes which regulates the conditions a user (who has a role) should meet, in order to access a resource. The access right refers to access modes such as read and write. The *isPermanent* attribute is used to determine if a policy is permanent or not. The policies are divided into four different categories. 'Authorization' policies define the actions that different roles are allowed to perform on resources. Role-Permission assignments and different contexts are used as conditions for making a decision in authorization policies. 'Delegation' policies are temporal policies under which users can delegate their access rights to other users under certain conditions specified by organizations's rules. The delegation validity duration, the portion of resource the grantee can access, the grantee access mode and whether the grantee can delegate his access right to other roles are defined in the *delegation policy* class. The *consent* class inherits all properties of the *delegation policy* class but it overwrites the type of grantee and grantor properties to persons, meaning that consent defines delegation between individual users instead of general roles.

'Refrain' policies revoke permission even if permission is given by other policies. 'Refrain' policies can be categorized into two types: *role refrain* that revokes permission from roles and *resource refrain* which avoids performing specific actions on a resource (e.g. avoiding remove action on a resource). 'Obligation' policies specify the action that must be performed when certain events occur. For example, security management policies specify what actions must be executed when security violations occur and who must execute those actions as well as what auditing and logging activities must be performed (when and by whom). These policies are explained further in Subsection 5.3.4.

The *user behavior* class uses the *context* class and audit trails for resources and users to extract required information to model the behavior of a user. The attributes of this class are the same as the tuple defined for modeling user behavior, explained in Section 4.1.

The *resource context* class considers the contexts over resources such as the access pattern made to the resource, the type of data the resource contains together with their sensitivity level and users who have previously accessed the resource. The sensitivity level can be defined based on factors such as the type of clinical data that the profile contains and users who have recently accessed the profile. For example the sensitivity level of a clinical profile containing cancer related data may be marked as high.

5.2.1 Context awareness

CAS offer several benefits in the healthcare environment such as authorizing users based on their context, adjusting security levels automatically and service sharing in a dynamic heterogenous environment [16, 50, 51, 57]. The blocks of the *decision making engine* layer, the main decision making blocks in our model, use concepts from CAS methods to model user behavior. Context aware models define logical constraints over context and restrict the set of possible context configurations. These constraints are placed in the *policy* class to maintain model integrity. The break glass procedure[7] is an essential requirement in the healthcare domain. CAS can be used to detect emergency situations by defining the occurrence of emergency situations in terms of existing contexts.

A major portion of the class diagram has been allocated to represent security related contexts. These contexts are inherited from the general *context* class (bottom of diagram) with detailed attributes used to express them. In the *team* class the following attributes are defined: validity period of team membership, criticality of the team operation, the

[7]Break glass refers to a quick means for a person who does not have access privileges to certain information, to gain access when necessary.

team members, whether there is any specific location assigned to the team and whether the team is temporary or permanent. The *location* class stores the emergency level of the target location (i.e. how critical is the health status of the patients hospitalized in that location) and provides a method for computing proximity of requester and resource. An additional context named *emergency*, which determines a situation's emergency level based on parameters such as time, location, role and resource, is defined under the class *context*. The class *user behavior*, composed of a set of contexts, represents the user behavior concept to make access control decisions. The *user behavior* class also contains additional information, explained in Section 4.1.

5.3 Proposed architecture

An architecture is designed to use the concepts introduced in the previous sections and deliver the desired access control functionality. The architecture is shown in Figure 5.2. The figure is divided into different layers that provide a high level categorization of different blocks (each box is called a block) of the model. The blocks of a layer perform a common major task.

The configuration of the system can be dynamically entered and stored via *input, representation* and *storage* layers. A database is included to store requesters data and security effective factors values. Once the database is filled with appropriate data by the security administrator, the generic model is configured for a specific system. Access requests are submitted to the model in the *connection point* layer and the final result is also returned to this layer as well. The *connection point* layer is a part of the communication bus of the distributed system. The model captures the context of the user who is making a request, checks for different policy rules, applies the results of behavior based constraints and returns the final access decision. In this section all blocks of the model are introduced and their responsibilities and specifications are explained.

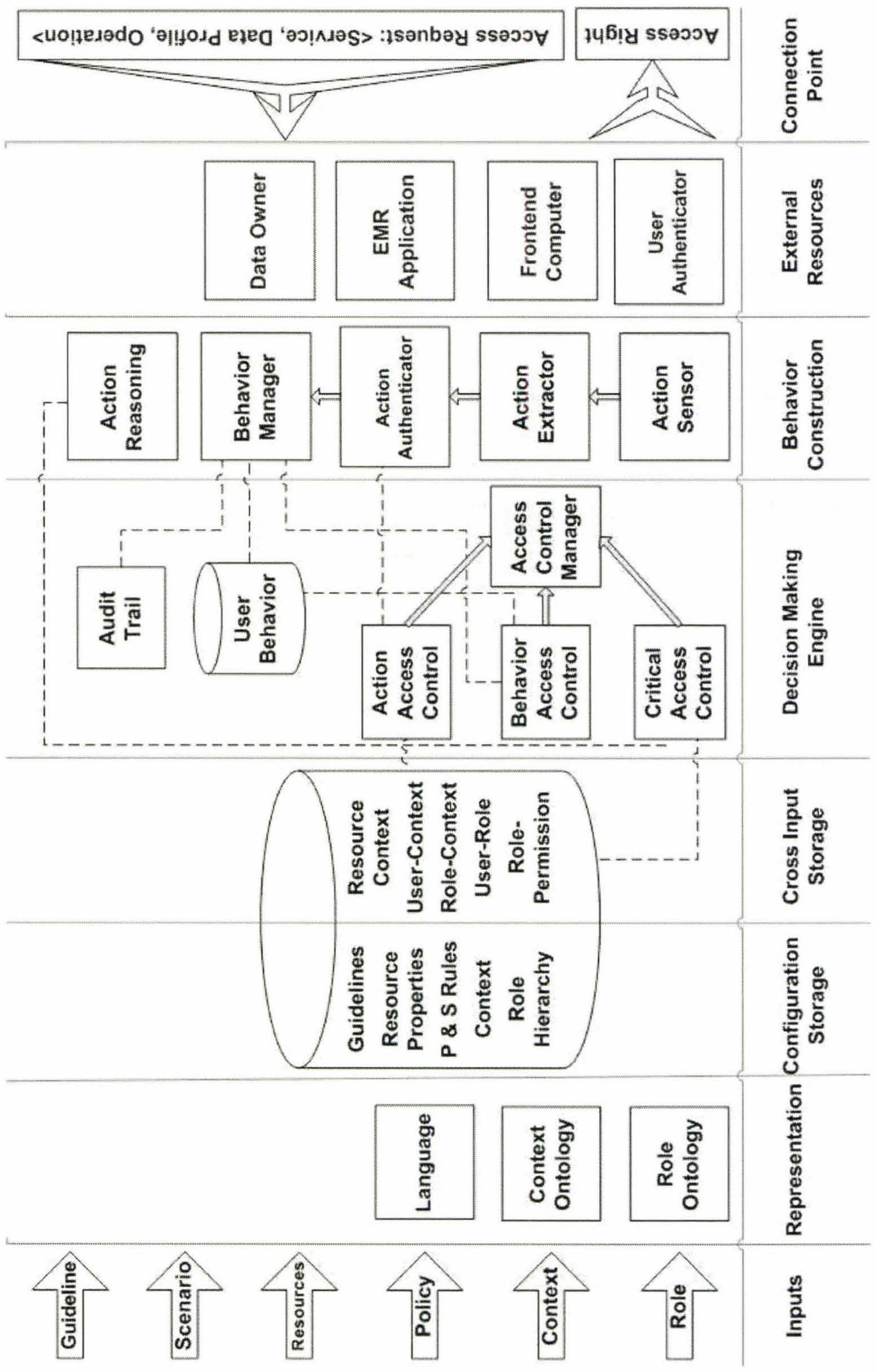


Figure 5.2: The architecture of the proposed access control model

5.3.1 Input

The inputs are the same as the security effective factors explained in Section 5.2.

5.3.2 Representation

In order to make the system interoperable and usable in different environments, we have to map input factors (from the *input* layer) to a common standard format. In this way when a workflow spans multiple organizations with different security architectures, no change to internal security architectures is required. In the healthcare domain, HL7 RIM provides a hierarchy for clinical roles which we adopt as our standard ontology [39]. Also the languages box supports common policy languages (such as Rei, Ponder, and X-GTRBAC) to facilitate interconnection with different systems. This box is specified in more detail in Section 7.2.

Role ontology in healthcare. The ontologies should be derived from standards in each domain. The security roles of healthcare providers are refined by HL7 as shown in Table 5.1. A similar hierarchy for the administrative side of patient care is also offered by HL7.

5.3.3 Storage

Configuration storage. Repositories reside between the *input* layer and the *decision making engine* layer as an interface for the engine. The purpose of using the repositories is to avoid losing model generality by making the engine independent of any special data format. The input data are stored in the repositories.

Cross input storage. This layer stores the relation between entities of the *configuration storage* layer such as association between users and roles. The dynamic attributes of system entities such as contexts of users and resources are also stored here.

Healthcare Roles	
Roles	Sub Roles
Audiologist	
Dental Hygienist	
Dentist	Dentist, Oral Surgeon
Dietitian	
NW Medicine Providers	Certified Acupuncturist (CA), Licensed Massage Therapist
Nurse	Clinical Nurse Specialist, Clinical Registered Nurse Anesthetist, Licensed Vocational Nurse, Nurse Midwife, Nurse Practitioner, Registered Nurse (RN)
Optometrist	
Pharmacist	Pharmacist-Apothecary, Pharmacist-Clinical
Physician	Chiropractor, DO/Osteopath, Homeopath, MD/Allopath, Naturopath, Pathologist, Podiatrist (DPM), Psychiatrist, Radiologist
Physician Assistant	
Psychologist	
Social Worker	
Speech Pathologist	
Technician	Cardiology Technician, Medical Laboratory Technician (MLT), Pharmacy Technician, Prosthetic Technician
Technologist	Cytotechnologist, Laboratory Technologist, Medical Technologist (MT), Radiologic Technologist
Therapist	Certified Educational Therapist, Kinesiotherapist, Occupational Therapist, Musical Therapist, Occupational Therapy Assistant, Physical Therapist, Physical Therapy Assistant, Recreational Therapist, Respiratory Therapist, Speech Therapist, Vocational Therapist
Veterinarian	

Table 5.1: Hierarchy of healthcare roles offered by HL7

5.3.4 Decision making engine

This layer uses the data gathered from other layers and makes the access control decision based on different factors. The access request and user data are passed to four blocks of this layer to get the final result. The blocks are now briefly introduced; detailed explanations will follow.

Critical Access Control enforces the privacy and policy rules including relations between users, roles, resources and permissions. This block is responsible for reasoning over different rules to discover the policy that should be applied for a user. *Action Access Control* checks for domain membership and CAAC constraints introduced as "single action" behavior in Section 4.1. *Behavior Access Control* checks for "daily behavior" defined in Section 4.1. This block compares the behavior of the user with the *common behavior*. *Common behaviors* are dynamically generated based on data analysis and new inquires are verified against them. The *Access Control Manager* manages the decision based on the results gained from the other access control blocks in the *decision making engine* layer. Different privacy and policy rules and user behavior constraints affect the access decision in different ways.

Since access control decisions are made in this layer, it is the best place to put the *Audit Trail* block. The audit trail establishes a historical record of user or system actions over a period of time and provides an answer to the question: "*what have you done?*". IHE has a refined list of audit trail events for distributed healthcare environments [49].

The *User behavior repository* is placed in this layer to record the frequency of information exchange between different blocks of this layer. Also, as some blocks from lower layers should access the *user behavior repository*, this repository could not be placed in any of the layers introduced before.

We now give the detailed specification of the major blocks of the *decision making engine* layer.

Action access control

The *single action* part of the behavior based access control explained in 4.1.2, is specified here. This block accepts an action tuple as input and checks if the attribute values of the action tuple have been previously used by this user. This checking is performed by running queries against the *user behavior repository*. The output of the block is a binary vector of membership of attribute values (i.e. 1 if the attribute value was used before and 0 otherwise). The number of 1s in the binary vector represents the portion of attribute values that are used before. Algorithm 3 specifies the described method. This algorithm accepts a user and his current action tuple as input and returns a vector of 0's and 1's associated with action attributes based on the membership of attributes values to values previously used in the user behavior history.

Behavior access control

This block considers the behavior in one day. If a sequence (such as a guideline) for behavior exists, it will be used otherwise a common sequence can be extracted by analyzing user records. These guidelines are usually organization specific and require analysis of organization workflows. For example in the nursing domain, the Canadian Nurses Association (CNA) [2] provides guidelines for Canadian nurses. They released a document called *Advanced nursing practice - a national framework* [23] which describes competencies and regulations for nurses.

Problem statement. Given a list of actions for a user, find the common sequences that happen for each attribute (if such sequences exist) to construct the user behavior.

Algorithm 4 is one solution to the stated problem. This algorithm has two functionalities: finding the common attribute values and determining the order of a sequence of attribute values. Step 8 can be performed as follows: given the first N extracted attribute values, the first element of the common sequence is the attribute value which appears most in the beginning of sequences of different days. Other positions (second to last) can be found in similar way, i.e. omitting the attribute values that are used in previous positions and choosing the attribute value with greatest occurrence.

Algorithm 4 can be run for sequences with different lengths and for different sets of attributes. Assuming that the attribute values are sorted from 1 to m by their occurrence in a list of action tuples, we can consider a window of length n ($n < m$) and move it over attribute values and run Algorithm 4 over the list of actions. Using the window approach is more useful for small N since sequences of short length do not provide much information. Ordering and transitivity rules can be used to construct longer sequences of attribute values. The attributes of the action tuple that we consider as input to the algorithm are role, location, team, patient profile or type of data. The *time* attribute can also be attached to attributes to obtain the occurrence time of sequence items in addition to their order.

The algorithms offered in this section aim at providing a minimum specification for the model. They can be extended and specified in more detail. Considering Algorithm 4, if a nurse follows two different workflows in two shifts, this algorithm just extracts the behavior of one of these days. This means that the *daily behavior* of one of the shifts does not match with the *common behavior*, which is not a good way to model user behavior. The algorithm can be extended in one of these ways to overcome this limitation: extract more than one sequence as the *common behavior* and compare the *daily behavior* with them to find a match or assign time stamps to the extracted *common behaviors* and compare the *daily behavior* with the *common behavior* within similar periods of time.

Whenever a new action tuple is input to the *behavior access control* block, this tuple together with tuples received from the beginning of the day (the *daily behavior*) are compared with the *common sequence* from Algorithm 4 which has the same length as the daily behavior. The number of common attribute values and their order determine the matching of two sequences. The outputs of the block are the portion of common attribute values (i.e. the number of common attribute values divided by the total number of attribute values) and the portion of correct ordering (i.e. the degree to which the orders are similar) as shown in Algorithm 5.

Critical access control

The policies introduced in Section 5.2 are specified in this block. A policy specification language is chosen that supports the desired policies. Using a language provides flexibility and eases maintenance compared to hard coding policies into the application. The drawback of course is the difficulty of expressing policies using a language in comparison to directly converting the logic into source code. Access requests are entered as inputs to this block and a rule based process is followed to gain the final result.

In order to obtain a better intuition about the policies, a general format for the different policies is now described.

- **Extended RBAC rules**, such as role r_i can perform operation op_j on resource res_k or can invoke service s_m under condition $cond_n$. A general instance of this rule is: physicians can update a patient's profile, if they are that patient's associated physician.

- **Delegation**, such as role r_i can delegate permissions p_j (i.e. performing operation op_k on resource res_m) to role r_n under condition $cond_q$ for period per_r. Patient Consent is considered as a delegation where the patient is the delegator who authorizes care givers to edit his profile. Generally the owner of a profile should be able to authorize others for manipulating his profile.

- **Team**, such as joining a team t_i as member r_j provides permissions p_k for a period per_m. For some teams, team members gather if certain preconditions (such as context constraints) are satisfied. For example an emergency team would form if an emergency situation happens.

- **Resource**, such as permission p_i is not allowed on resource res_j under condition $cond_k$. This type of policy can be modeled by RBAC with some overhead (i.e. adding access constraints to all roles). However there might be general constraints over resources independent of users and roles (for example a resource can not be deleted).

- **Refrain policies**, such as refraining permission p_i from role r_j in condition $cond_k$. For example a role is not allowed to modify a profile when he is not in a certain location or he is not assigned to the profile. This kind of policy can be specified using extended RBAC, i.e. adding conditions to the RBAC.

- **Obligation policies**, such as auditing special control events to execute security administrative actions.

- **Context**, such as constraints that are used in different situations. For example a permission is allowed under a special context constraint, i.e. the requester should be in a special location-time setting. Another context is the *resource context* (introduced in Section 5.2) which is used to check the eligibility of the requester based on the existing information about the contexts of a resource. In other words we check if the requester and his contexts matches with the resource context. The *resource context* contains two types of attributes: one type describes the resource (such as sensitivity of the resource and the type of data the resource contains) and the other type considers the requester's interaction with the resource (such as access pattern). Therefore some attributes should be accordingly assigned to the requester to compare them with resource attributes. For example equivalent to the 'type of data' that a resource has, a 'typical kind of data' is assigned to the user and these two sets are compared to compute the overlap (an eye specialist usually looks at patients with eye problems).

In Section 7.2 two policy specification languages and their supporting packages are introduced. The policies mentioned above will be specified in that section.

Access control manager

This block is responsible for determining the access decision based on the results gained from the other access control blocks. Both the *action access control* and *behavior access control* blocks run their algorithms separately for the user and role. The user's role should be exact and specific (radiologist is specific compared to physician, as the physician role involves a broad range of roles) to get reasonable and useful results from the action and behavior access control blocks.

Problem statement. Given the results of the security and privacy rules, action, and behavior analysis, find an algorithm that determines the final access right for a request. A major decision to make is solving the conflict between the results of different blocks.

Since the privacy and security rules are based on legal obligations in organizations and between organizations, any decision made by the *access control manager* block should not break these rules. That is the results of the action and behavior access control blocks are not able to change the result of security and privacy rules. Algorithm 6 offers an algorithm that respects the above constraint.

The user's credit schema is stored in the *access control manager* block, since this is the only block which is aware of the results of other blocks. The credit depends on whether action and behavior satisfy the threshold and whether conflicts occur between the security and privacy and the behavior. In Algorithm 6, whenever the required threshold for behavior is satisfied, *behaviorCredit* is increased by 1 and if not satisfied, it is decreased by 1. *criticalCredit* stores the compatibility of behavior and critical access controls, i.e. if the results of critical and behavior access control are the same, *criticalCredit* is increased by 1 and otherwise it is decreased by 1.

A question arises on how to determine the *threshold* variable of the algorithm. In other words, what is the threshold for considering a behavior as following *common behavior*?

Some possible answers are: allowing the security administrator to pass the value as a parameter to the algorithm; the value of the *threshold* attribute can be determined by the target *common behavior*, based on the extent to which a given behavior is required to match with the *common behavior*; calculating the average value achieved by users of the same role from their behavior history; the healthcare professionals can suggest the value based on their experience and clinical guidelines.

Audit trail

Different events that happen in our model are stored in the *audit trail* block to be able to analyze them to determine the reason for a failure or access violation. The IHE project has suggested a list of auditable events for healthcare applications including: node-authentication-failure, order-record-event, patient-care-assignment, patient-record-event, PHI-export, PHI-import, procedure-record-event, query-information, security-alert, user-authentication [49]. Access requests and the results returned by the model are stored in the audit trail block.

5.3.5 Behavior construction

This layer is responsible for constructing basic data for the *decision making engine* layer, i.e. the *action* and *behavior* concepts. Different blocks are required to capture and represent these concepts. *Action sensor* senses any changes in the attributes of the action tuple and informs another block to extract the required data. *Action extractor* composes the action tuple based on the data sensed by *action sensor*.

Action authenticator authenticates the context itself. Different methods such as: statistical analysis, distributed reputation, and confidence value, are used for authenticating contexts [69] (mostly for physical sensors). Attribute values are checked for validity, that is if the values are complete and correct. Erroneous values for critical attributes (such as person, role, team, requested profile and requested service) are rejected to maintain data consistency. For example one way to identify a location is using the Media Access

Control (MAC) address of network adaptors of computers found in that location. The *action authenticator* block is responsible for checking the validity of the MAC address. It also checks if the MAC address is mapped to the specified location. Also, if there is any association between attributes, their values are checked. For example the *action authenticator* block checks that a team session takes place in a specific location.

Behavior manager composes the behavior based on the input action tuple and the past history of user behavior and updates the *user behavior repository*. The *action reasoning* block uses context inference rules, that are input through the *input* layer, to infer the contexts that can not be directly sensed. For example, detecting emergency situations is a context which requires aggregation and reasoning over multiple contexts. These data are passed to the *critical access control* block to apply relevant rules.

5.4 Usage scenario

A simple scenario is described to navigate through the model blocks to determine how the model handles an actual request. The scenario is as follows: *Jane, a nurse in the Diabetes Department, requests to review the disease related data from the profile of Nero, a patient hospitalized in the same department.*

The *action sensor* block collects data from various sources. Jane's identity is passed from the *user authenticator* block, Jane's active role is obtained from the current session, locations are equivalent to network adaptor physical addresses of the *frontend computer* which is employed, Jane's team is provided by the *Electronic Medical Record (EMR) application* she is using and other attributes are extracted both from the request that Jane has made and from Nero's profile. Once these data are extracted, the *action extractor* creates the following action tuple:

<Jane, nurse, diabetes nursing station, shared health record, 2008.2.23-13:00, diabetes dep. nursing, null, Nero's diabetes data, review, diabetic information, null>

Action authenticator checks critical attributes (i.e. person, role, team, requested profile, and requested service) for valid values. For example the context repository is queried to see if *diabetes dep. nursing* is a valid active team or not. Also if Global Positioning System (GPS) is used to provide location information, hardware related authentications would be applied.

Behavior manager stores the action tuple in the *user behavior repository* and extracts Jane's activities for her current working day and passes it to the *behavior access control* block.

Suppose the actions Jane has taken today are as follows: Jane arrives to the diabetes department and logs into her account (supported by the hospital's EMR application) in the morning. She then checks the patients in the department, starting with Nadia. Jane reviews Nadia's profile and decides to order lab tests. Jane also administers medication to Nadia and updates the drug section of Nadia's profile accordingly. Around 11:00, Jane is asked to go to the emergency department. Therefore she delegates her privileges to Julie and leaves for the emergency department. During the emergency process, she retrieves allergy information about a newly arrived patient. Jane comes back to the diabetes department at 13:00 and continues to check patients. The user behavior repository data retrieved by *behavior manager* looks like:

<Jane, nurse, diabetes nursing station, EMR server, 2008.2.23-9:00, null, null, Jane's account, modify, reminders.requests.medical knowledge, login>

<Jane, nurse, diabetes nursing station, shared health record, 2008.2.23-10:00, diabetes dep. nursing, null, Nadia's diabetes data, review, diabetic information, null>

<Jane, nurse, diabetes nursing station, laboratory server, 2008.2.23-10:15, diabetes dep. nursing, null, Lab database.Nadia's lab requests, order lab test for Nadia, diabetic information, null>

<Jane, nurse, diabetes nursing station, shared health record, 2008.2.23-10:30, diabetes dep. nursing, null, Nadia's drug information, X drug injected, diabetic information, null>

<Jane, nurse, diabetes nursing station, role repository, 2008.2.23-11:00, diabetes dep. nursing, delegates to Julie for 3 hours, active nurses database, null, null, null>

<Jane, nurse, emergency station, shared health record, 2008.2.23-11:20, emergency team, null, Nancy's profile, get allergies, allergy data, null>

Action access control checks the membership of different attributes. Based on her behavior history, this block says that Jane has been in nursing, emergency, surgery and research teams. She has also accessed the profiles of Nadia, Nancy and Nero as her patients. If we follow the vector output described in the *action access control* of Subsection 5.3.4, the output will be composed of all 1s since all of the attributes have been used before.

Using Algorithm 4, *behavior access control* extracts Jane's behavior. For example Jane sequentially goes to the diabetes department, surgery, diabetes department and finally the library. Her presence in emergency does not follow a specific pattern. Algorithm 5 is then used to find the matching between Jane's behavior today with her past behavior - 1 is returned as the result represents a full match.

In the *critical access control* block, the user-role assignment and role-permission assignment are checked to identify if Jane has access to Nero's profile (and generally her role as a nurse is compared to the patient's role). Then *diabetes dep. nursing* team access rights state that she can access patient profiles within the diabetes department and restricts access to profiles of other departments. The delegation rule is not applied here since Jane is using her original rights. For the case that Jane delegated her privileges to Julie, the delegation rule checks if Jane and Julie are valid nodes for this delegation relation. If the relation is valid, the patients associated with Jane are assigned to Julie for a certain period of time. Context aware access control checks that Jane is accessing an internal resource of the department at a reasonable time of the day. It is also checked if the nurse associated to Nero (for a period of two weeks) is Jane. Nero's resource context lists the care givers who have recently accessed his profile, including Jane.

Having the results of previous blocks, the *access control manager* makes the final decision about Jane's request. Following Algorithm 6, *criticalCheck* remains true, since Jane is authorized by role checking. *Action access control* returned a vector of 1s and *behavior access control* returned 1 which means Jane passed the threshold for following a *common behavior*. Therefore the algorithm returns *access granted* as the final decision for this request.

Chapter 6

Access control model evaluation

In this chapter we discuss the model from different perspectives to evaluate its applicability, performance and characteristics. In Section 6.1, we discuss how the model satisfies a reasonable number of requirements for distributed environments. Then in Section 6.2, we discuss how the model collects required information from the target environment. A brief discussion on verifying the results returned by the model is offered in Section 6.3. In Section 6.4 an approach is given to deploy and test the system with test data. Finally, the proposed model is compared to existing access control methods in Section 6.5.

6.1 Requirements satisfaction

Here we revisit a number of requirements for access control methods in distributed environments (discussed in Section 5.1) and briefly describe how the proposed model satisfies them.

Organization Specific Privacy Rules. Each organization is able to enter its desired privacy rules through the privacy input. Unlike roles, no ontology is offered for modeling privacy in the healthcare domain. Therefore the proposed model provides a common data model for different types of policies and privacies and uses a specification language to express them. The *decision making engine* directly uses these rules.

Context Awareness. The blocks of the generic architecture of context aware systems are scattered through the model architecture. *Action sensor* acts as the sensor interface, *action extractor* retrieves the data, a part of the *behavior manager* block preprocesses the sensed contexts, the contexts are stored in a special format in the user behavior repository, and finally the application layer is embedded in the *behavior access control* and *action access control* blocks where the contexts are used for making access control decisions. It should be highlighted that the blocks of the model provide additional functionality compared to the CAS blocks explained in Section 5.3.

Sequence Control. Context awareness of the model provides the required run time information necessary to create a sequence of different attributes. The *behavior access control* block uses the sequence of actions that are performed by a user to make the access control decision.

Generality. Different security effective factors in distributed environments are recognized and represented in a class diagram. Also the model architecture provides ontologies and common modeling languages to cover the wide and varying range of requirements and specifications. These facilities provide an interoperable framework to enter effective factors to the system which are used in other blocks of the model.

Decentralization. The model trusts the communication bus capabilities to provide access to different resources. The model does not include this service to avoid dependency on a special platform or architecture.

Modification of Policies. The *decision making engine* uses the policy repositories to retrieve the organizational policies, instead of using hard coded policies. Therefore security administrators of organizations can modify and update these repositories via the input facility of the model.

Temporal Roles. The *policy* class in the proposed class diagram for security effective factors considers policies which are valid for a certain period of time. In particular, delegation policies consider a validity period for the delegation relation.

Auditing. The *audit trail* block records all of the requests, inputs and decisions that occur in different blocks of the model.

6.2 Collection of action attributes

The model collects a significant amount of information per request. It is not desirable to force users to enter any additional data above that required for their normal task workflows. Hence there is a concern about how much additional data the user should provide and how to proceed if some data are missing or invalid.

Considering the attributes of the action tuple, most of them can be derived automatically from the user context. User identity is derived from the authentication block; location and time are provided by the front end computer employed by the user; the type of service and data are extracted from the user access request; login/out events are detected by the EMR available on the frontend computer. The user should notify the system whenever he joins or leaves a team. Therefore the user is not required to provide that much additional data for the attributes of an action tuple.

In case of missing attributes, if it is not possible to use default values, the user behavior can not be extracted and the functionality of the model would be equivalent to traditional access control methods (i.e. following static rules). However if the model is used in an environment for a certain period of time, the required knowledge about different entities would be stored in repositories, thus greatly reducing the possibility of the occurrence of missing attributes in action tuples. The validity of attributes (e.g. being from a valid domain) is considered in the *action authenticator* and *action access control* blocks.

6.3 Validity of decisions

The first and most important concern about an access control method is whether it is working or not. This concern is emphasized in the proposed model since it might not always be possible to accurately model and reason about a user's behavior. Is it possible that a user is mistakenly granted access because of the credits he has for his good behavior?

The access control decision is a two stage process. The first stage is based on the different concepts introduced in robust traditional access control methods. This stage employs pure rules and is independent of the concepts introduced for *user behavior*. The second stage extracts the *user behavior* and applies it for access control. This stage is not purely based on behavior analysis and always the guidelines and organizational workflows are used to adjust the *common behavior*. If a user fails to pass the first stage, the other credits that he gains for his behavior can not change the access control decision.

6.4 Testing

Initially, the repositories related to user behavior and the usage context of resources do not contain any data. Once the model is deployed and used in an environment, the environment specific data would be stored in these repositories. Therefore the testing process should only begin after there is enough information in the repositories. Determining the minimum amount of collected information requires further analysis and is left as future work.

As an evaluation process, users with different but known behaviors should access different resources and the extracted *common behavior* must be compared with the known user behavior. In this way the accuracy of the model to correctly visualize the behavior of the user is examined. A method should be introduced on how to evaluate whether the common behavior matches the actual behavior of the user. There might be situations where the results of behavior analysis and the results of evaluation against privacy and security rules, do not match. These situations should be evaluated as a means for improving the behavior analysis of the model.

In order to test the correctness of the specified privacy and security rules, the requests should be broad enough to test different methods of authorization. They must cover privileges allowed based on team membership, delegation, and different permissions (specific and generic).

6.5 Comparison with other methods

In this section the proposed model is compared with several related access control methods. In Table 6.1, the criteria that should be considered for the access control methods are listed in the left column. These criteria are extracted from the requirements mentioned in Section 5.1. The numbers in the third row are the bibliographical citations of the methods which are being compared. The second row provides a category for these methods. In the table, *y* means that the method satisfies the criteria, *n* means that the method does not satisfy the criteria, and *n/a* means that the criteria does not apply to the method (when a criteria which is specific to the healthcare domain is applied to a generic access control method, this notation is used). It is seen that the proposed model satisfies all of the requirements.

Access control methods comparison															
	TBAC	CAAC							RBAC				CSAC	SAAC	Proposed Model
Factors	[33]	[41]	[66]	[67]	[60]	[53]	[17]	[71]	[54]	[55]	[19]	[72]	[42]	[70]	Proposed Model
Context-awareness	y	y	y	y	y	y	y	y	n	n	n	n	y	y	y
Dynamicity	y	y	y	y	y	y	y	y	y	n	n	n	y	y	y
Delegation	n	n	n	n	n	n	n	n	y	n	n	y	n	n	y
Standards compatibility	n/a	n	n/a	n	y	n/a	n/a	n/a	n/a	n	y	n	n/a	y	y
Semantic interoperability	n	n	y	n	n	y	n	n	n	y	y	n	n	n	y
Emergency handling	n/a	n	n/a	y	n	n/a	n/a	n/a	n/a	n	n	n	n/a	n/a	y
Audit trail	n	n	n	n	n	n	n	n	n	n	n	n	y	n	y
Formal definition	y	y	y	n	n	n	y	y	y	n	y	y	n	y	y
Policy flexibility	n	y	y	n	y	n	n	n	y	n	y	y	n	y	y
Visualization	n	n	n	n	n	n	n	y	y	n	n	n	y	y	y
Implementation	y	y	n	y	n	y	y	y	n	y	y	y	y	y	y

Table 6.1: Comparison of different access control methods

Chapter 7

Simulated Environment

In this chapter we provide an implementation for the architecture of the proposed access control model which was specified in Chapter 5. We introduce a simulated environment as the prototype for the proposed architecture. This implementation aims to provide a prototype for different sections of the architecture and also represents the major characteristics of the model. In Section 7.1 the ontology described in the *representation* layer of the model architecture (see Section 5.3) is explained. This ontology is modeled and visualized using available technologies and tools. In Section 7.2 we consider the interactive entry of privacy and policy rules into the system. We show how to use an existing language to specify these rules. The required scanner, parser, and the reasoning engine for that specification language are explained in Section 7.3.

The algorithms required to analyze the data to extract behavior (see Section 5.3.4), are implemented and discussed in Section 7.3. Different blocks of the *decision making engine* layer are created and used to process a request. The implemented blocks are tested with simulated data and the results are shown in Section 7.4.

The standard three layer architecture is used for this prototype. It includes the *data access* layer which interacts with the higher layers and the database and acts as an interface for the database. The *business layer* contains the core logic. Finally, the

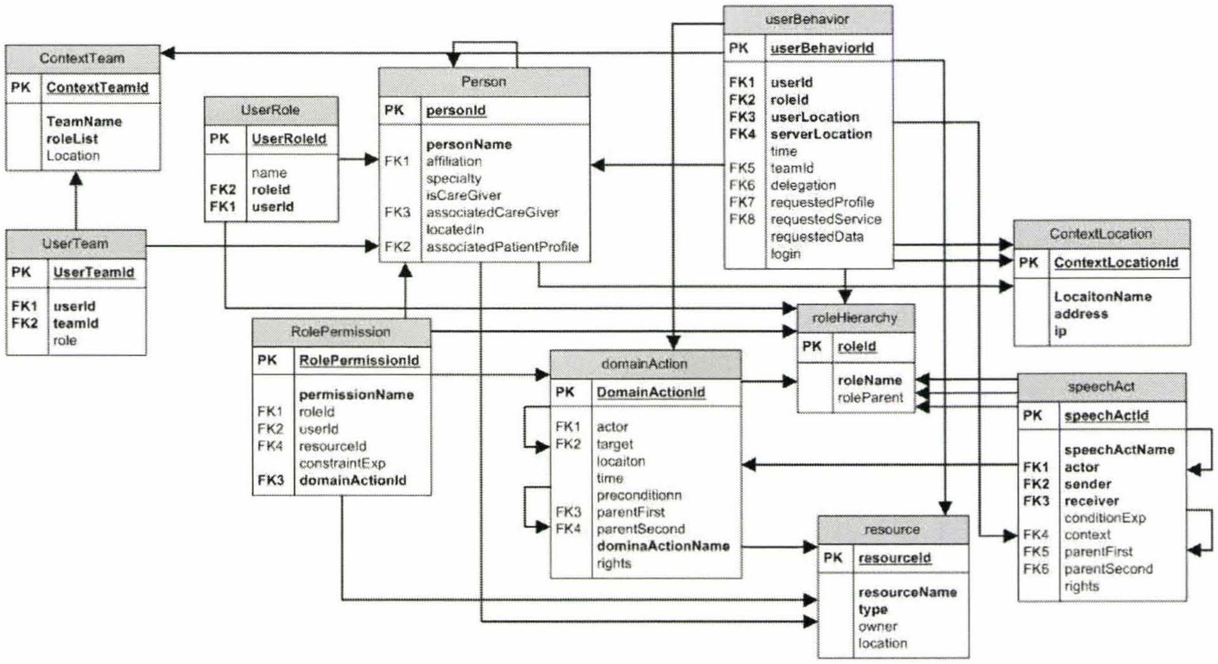


Figure 7.1: The relational schema of the database

presentation layer offers an interface to the users. The following technologies and tools are used: Java, Eclipse, and MySql database.

The relational schema of the database is shown in Figure 7.1. The actual database is created with the MySql Database Management System (DBMS) and is tested with sample data to assure delivery of desired functionality and consistency of the database design. The tables and their fields are shown in Figure 7.1. The primary key of each table is marked by *PK*, foreign keys are marked by *FK* and NOT NULL fields are indicated in **bold**.

Sample data are generated in a way that they can test the modification of foreign-primary key relationships between tables. For example when a record is entered to the *UserRole* table, *Person* and *roleHierarchy* tables are checked respectively to find the associated values for the *roleId* and *userId* attributes.

7.1 Ontology specification

In this section we first specify the ontology that was offered for the *representation* layer, using OWL. Then we use an ontology editor to visualize the OWL file. OWL is an extension to the Resource Description Framework (RDF) which employs XML to represent ontologies. OWL is suitable for both presenting information and processing the content of information.

The Role hierarchy and context classes of the class diagram are represented by OWL. Automatic approaches are suggested [32, 68] to convert UML diagrams to OWL files that can be used for feature flexibility (i.e. changes in UML diagrams would result in changes in the associated OWL file). OWL is also used to map between specific system hierarchies and the hierarchy ontology offered by the model. The *owl:equivalent* feature is employed to define equivalency between attributes of hierarchies. Also, contexts and their relations are modeled using the *owl:objectProperty* and *owl:dataTypeProprty* features of OWL. In the following a small portion of a context OWL file expressing general location and its usage as the location of an emergency situation are represented.

```
<Declaration>
    <OWLClass URI="&Ontology1203016001343;Location"/>
</Declaration>
<SubClassOf>
    <OWLClass URI="&Ontology1203016001343;Location"/>
    <OWLClass URI="&Ontology1203016001343;Context"/>
</SubClassOf>

<Declaration>
    <DataProperty URI="&Ontology1203016001343;Location_Value"/>
</Declaration>
<DataPropertyDomain>
    <DataProperty URI="&Ontology1203016001343;Location_Value"/>
    <OWLClass URI="&Ontology1203016001343;Location"/>
```

```
</DataPropertyDomain>
<DataPropertyRange>
    <DataProperty URI="&Ontology1203016001343;Location_Value"/>
    <Datatype URI="&xsd;string"/>
</DataPropertyRange>

<Declaration>
    <ObjectProperty URI="&Ontology1203016001343;Emergency_Location"/>
</Declaration>
<ObjectPropertyDomain>
    <ObjectProperty URI="&Ontology1203016001343;Emergency_Location"/>
    <OWLClass URI="&Ontology1203016001343;Emergency"/>
</ObjectPropertyDomain>
<ObjectPropertyRange>
    <ObjectProperty URI="&Ontology1203016001343;Emergency_Location"/>
    <OWLClass URI="&Ontology1203016001343;Location"/>
</ObjectPropertyRange>
<FunctionalObjectProperty>
    <ObjectProperty URI="&Ontology1203016001343;Emergency_Location"/>
</FunctionalObjectProperty>
```

Protege [10] is an open source ontology editor and knowledge-base framework that supports OWL. Protege offers graphical interfaces for creating OWL files. Protege also has a plug-in that visualizes (in graph format) the entities and the relations between entities defined in an OWL file. TGViz, a Protege plug-in, uses the TouchGraph package [12], a visualization solution, to visualize the relations. Figure 7.2 is a snapshot of the role hierarchy OWL file visualized by the TGViz plug-in of Protege.

7.2 Policy specification languages

Several policy specification languages can be used. Amongst these languages, Ponder [9]

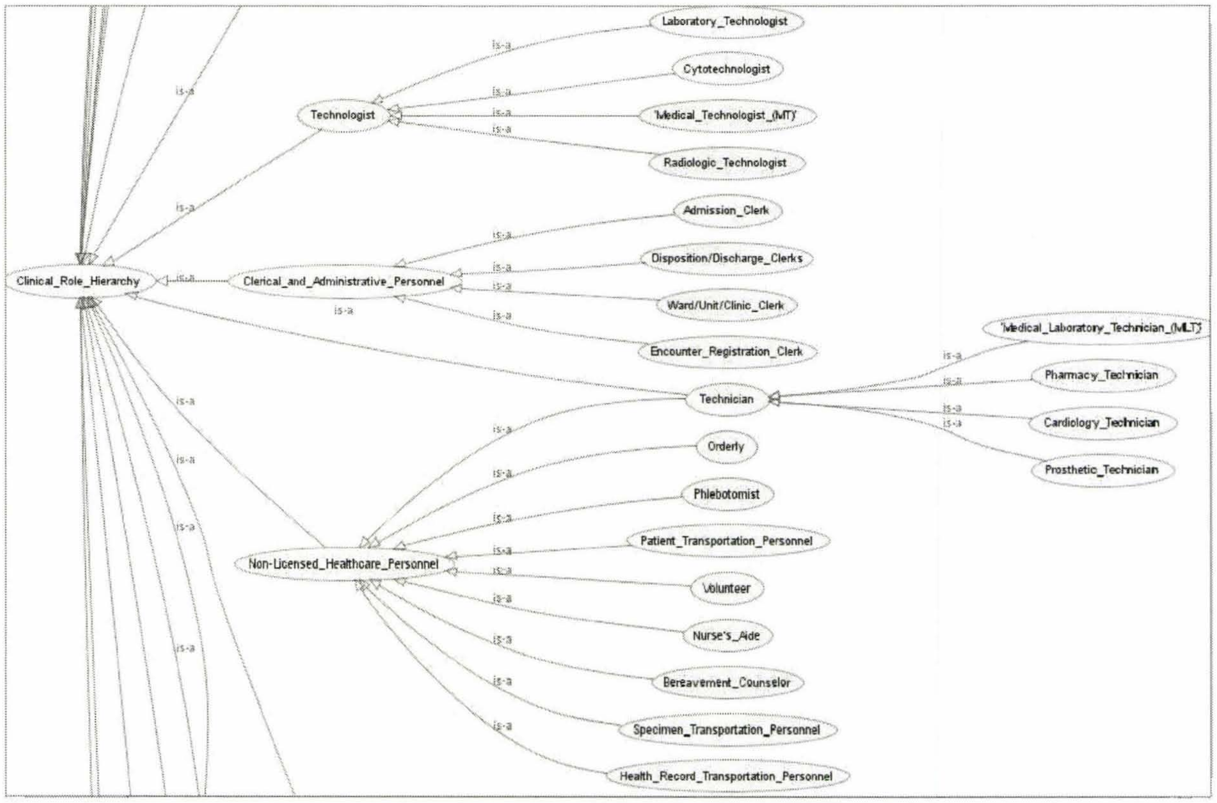

Laboratory_Technologist
Cytotechnologist
Medical_Technologist_(MT)
Radiologic_Technologist
Technologist
Admission_Clerk
Disposition/Discharge_Clerks
Ward/Unit/Clinic_Clerk
Encounter_Registration_Clerk
Clinical_Role_Hierarchy
Clerical_and_Administrative_Personnel
Medical_Laboratory_Technician_(MLT)
Pharmacy_Technician
Cardiology_Technician
Prosthetic_Technician
Technician
Orderly
Phlebotomist
Patient_Transportation_Personnel
Volunteer
Nurse's_Aide
Bereavement_Counselor
Specimen_Transportation_Personnel
Health_Record_Transportation_Personnel
Non-Licensed_Healthcare_Personnel
is-a

Figure 7.2: Role hierarchy modeled by OWL and visualized by Protege

and Rei [11] are two languages that support the requirements of our model. These requirements include checking role based constraints, team based constraints and context aware constraints. In this section we briefly introduce these two languages and describe how to specify policies with the Rei language.

7.2.1 Ponder

The Ponder language provides a common means of specifying security policies that map onto various access control implementation mechanisms for firewalls, operating systems, databases and Java. It supports obligation policies that are event triggered condition-action rules for policy based management of networks and distributed systems. Ponder is declarative, strongly-typed and object-oriented which makes the language flexible, extensible and adaptable to a wide range of management requirements [26].

Ponder2 is a tool that supports different types of policies offered by the Ponder language. Ponder2 comprises a self-contained, stand-alone, general-purpose object management system with message passing between objects. It incorporates an awareness of events and policies and implements a policy execution framework. Ponder2 implements a Self-Managed Cell (SMC) which is defined as a set of hardware and software components forming an administrative domain that is able to function autonomously. Management services interact with each other through asynchronous events propagated through a content-based event bus. Policies provide local closed-loop adaptation, managed objects generate events, and policies respond to and perform management activities on the same set of managed objects. Ponder2 has a high-level configuration and control language called PonderTalk and user-extensible managed objects are programmed in Java. PonderTalk is a high-level language, based on Smalltalk, that is used to control and interact with the Ponder2 SMC.

Regarding applicability and ease of use of this tool, the product suffers from a lack of realistic samples that can be followed or extended to develop policies for real world cases. Also the language used to implement the policies (PonderTalk), has a complex syntax for expressing complicated policies. There is not enough documentation on how to integrate the engine of Ponder2 with legacy applications and Java APIs. Regarding these issues, learning how to use the software requires an unreasonable amount of time.

7.2.2 Rei

Rei is a policy language based on OWL that allows policies to be specified as constraints over allowable and obligated actions on resources in the environment. Rei includes meta policy specifications for conflict resolution, speech acts for remote policy management and policy analysis specifications like what-if analysis and use-case management, making it a suitable candidate for adaptable security in the environments under consideration.

The Rei engine, developed in XSB, reasons over Rei policies and domain knowledge in RDF and OWL to provide answers about the current permissions and obligations of an entity, which are used to guide the entity's behavior. In order to be able to install the Rei interface, four other packages should be installed on top of each other. The first layer is XSB, then FLORA, then FOWL, then YAJXB and finally the Rei interface. These packages have compatibility issues. With the latest Linux C/C++ compilers, it is impossible to run the software.

7.2.3 Specification with languages

Considering the issues with both Ponder and Rei, it was decided to use Rei to specify policies but to not use the available tools. The advantage of Rei is that using OWL and ontologies eases semantic interoperability, an important requirement of the proposed model. Regarding the difficulties of using available tools, we implemented an engine for interpreting and applying the Rei specifications, explained in Section 7.3.

Rei is composed of several ontologies, as represented in Table 7.1. Each ontology describes the classes and properties associated with that domain and uses the unique XML namespace of that ontology. The description of ontologies can be found in Table 7.2.

The policy specification is divided into separate files, each describing a different portion of the policy specification.

Ontology file. Different entities, contexts, and the general concepts (classes) of Figure 5.1, are introduced in an ontology file. This operation is straightforward as the class diagram has previously been expressed in OWL notation. As an example, some of the entries of the ontology file (in the form *entity(attribute)*) are: Person(name, affiliation, isCareGiver), Patient (locatedIn, associatedCareGiver), places and actions.

Instance file. Instances of the classes in the ontology file are expressed in an instance file. This file also contains class definitions that are subclasses of classes introduced in

Rei Ontologies	
Prefix	Class list
policy	Class : ReiRoot, Policy, Granting
metapolicy	Class : MetaPolicy, ModalityPrecedence, Behaviour, MetaMetaPolicy, Priority Subclass of Priority : RulePriority, PolicyPriority
entity	Class : Entity Subclass of Entity : Agent, Object, Variable
deontic	Class : DeonticObject Subclass of DeonticObject : Permission, Obligation, Prohibition, Dispensation
action	Class : Action Subclass of Action : DomainAction, SpeechAct Subclass of SpeechAct : Delegation, Revocation, Obligation, Dispensation
constraint	Class : Constraint Subclass of Constraint : SimpleConstraint, BooleanConstraint Subclass of BooleanConstraint : And, Or, Not
analysis	Class : Analysis Subclass of Analysis : WhatIf, UseCase Subclass of WhatIf : WhatIfProperty, WhatIfPolicyRule Subclass of UseCase : StatementUseCase, DeonticUseCase

Table 7.1: Rei ontologies class list

Rei Ontologies Description	
Ontology	Description
policy	Policies are used to guide the behavior of entities in the policy domain. A policy primarily includes a list of rules and a context used to define the policy domain. It could also include a list of defaults used to interpret the policy, and a set of conflict resolution specifications. A granting associates a set of constraints with a deontic object to form a policy rule. This allows reuse of deontic objects in different policies with different constraints.
metapolicy	Rei specifications include metapolicy constructs for how policies are interpreted and how conflicts can be resolved. Rei models two main types of meta policies: (1) for defaults and (2) for conflict resolution to handle different policy requirements. Meta policies for defaults include behavior and meta-meta policies and meta policies for conflict resolution include priorities and modality precedence.
entity	Any human user, software agent or hardware resource is described as an entity:Entity. Currently, the Entity class has only one property, entity:affiliation, which is used to specify what organization an entity belongs to.
deontic	This class is used to create permissions, prohibitions, obligations and dispensations over entities in the policy domain. It includes constructs for describing what action the deontic is described over, who the potential actor (or set of actors) of the action is and under what conditions is the deontic object applicable.
action	This ontology is one of the most important in the Rei specifications as policies are described over possible actions in the domain. This class includes properties that are required for all actions. Though the execution of actions is outside the policy engine, Rei includes a representation of actions that allows more contextual information to be captured and allows greater understanding of the action and its parameters. It also permits domain dependence for information about actions to be added.
constraint	A constraint is used to define a set of objects like a set of graduate students, or a set of actions whose targets are laser printers. There are two subclasses of constraints: SimpleConstraint and BooleanConstraint.
analysis	To enable the development of consistent and valid policies, Rei provides two specifications: use-case management and what-if analysis.

Table 7.2: Rei ontologies description

the ontology file and have constraints over entries of the instance file. In the following example an action called *NursDiabAction* is shown that represents those actions that a nurse of the diabetes department performs.

```
<owl:Class rdf:ID="NursDiabAction">
    <rdfs:subClassOf rdf:resource="DiabAction"/>
    <rdfs:subClassOf rdf:resource="MajorProfileChange"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="actor"/>
            <owl:allValuesFrom rdf:resource="Nurse" />
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Other entries of the instance file are direct instances of defined classes. The following example represents how to introduce a patient to the system:

```
<hosp:Patient rdf:ID="Nero">
    <hosp:associatedCareGiver rdf:resource="Jane"/>
    <hosp:affiliation rdf:resource="DiabetesDept"/>
    <hosp:PatientProfile rdf:resource="NeroEHR"/>
    <hosp:locatedIn rdf:resource="room223"/>
</hosp:Patient>
```

Policy file. Policy files contain constraints, different types of policies and meta policies and the collection of active policies. Constraints are logical expressions restricting the attribute values of objects. The following example describes the constraint of someone being a member of the diabetes department.

```
<constraint:SimpleConstraint rdf:ID="IsMemberOfDiab">
    <constraint:subject rdf:resource="var1"/>
    <constraint:predicate rdf:resource="affiliation"/>
    <constraint:object rdf:resource="DiabetesDept"/>
    <policy:desc>All members of diabetes Department</policy:desc>
</constraint:SimpleConstraint >
```

Policies are described using *permission/prohibition*, *delegation/revocation* and *obligation* tags. Different policies (role, team, context) are modeled based on the constraints that must be checked prior to giving permission to a user. The policies can be specific or generic. The former assigns access rights to individuals while the latter describes the general conditions under which an access right is granted or denied. An example of giving specific permission to Jane, a nurse of the diabetes department, to perform an action defined by nurses in the diabetes department is shown below:

```
<deontic:Permission rdf:ID="Perm_Jane">
    <deontic:actor rdf:resource="&inst;Jane"/>
    <deontic:action rdf:resource="&inst;ANursDiabAction"/>
</deontic:Permission>
```

The following example is a generic delegation rule that allows Jane to delegate nurse actions in the diabetes department to other members of the department. The tag *<constraint:And>* is called a boolean constraint and performs the logical 'and' operation on expressions.

```
<action:Delegation rdf:ID="JaneToDiabetesMembers">
    <action:sender rdf:resource="&inst;Jane"/>
    <action:receiver rdf:resource="&deptpolicy;var1"/>
    <action:content>
        <deontic:Permission>
            <deontic:actor rdf:resource="&deptpolicy;var1"/>
            <deontic:action rdf:resource="&deptpolicy;var2"/>
        </deontic:Permission>
    </action:content>
    <action:condition>
```

```
        <constraint:And>
            <constraint:first rdf:resource="&deptpolicy;IsMemberOfDiab"/>
            <constraint:second rdf:resource="&deptpolicy;IsDiabNurseAction"/>
        </constraint:And>
    </action:condition>
</action:Delegation>
```

The following example expresses a generic obligation policy that nurses must sign up as responsible nurse before starting their shifts.

```
<deontic:Obligation rdf:ID="Obl_NurseSigningUp">
    <deontic:actor rdf:resource="var1"/>
    <deontic:action rdf:resource="&inst;SigningUp"/>
    <deontic:startingConstraint rdf:resource="IsNurse"/>
    <deontic:endingConstraint rdf:resource="HasShiftStarted"/>
</deontic:Obligation>
```

Meta policies define the priority between different rules/policies; default policies; explicit permission or prohibition. The following example is a meta policy stating that the emergency policy has a higher priority than the diabetes department policy. This means that if an emergency situation is detected, the emergency policies must be applied and the normal policy of the diabetes department can be ignored.

```
<metapolicy:PolicyPriority rdf:ID="DiabPolicyGreaterPriority">
    <metapolicy:policyOfGreaterPriority rdf:resource="&deptpolicy;DiabEmrgPolicy"/>
    <metapolicy:policyOfLesserPriority rdf:resource="&deptpolicy;DiabPolicy"/>
</metapolicy:PolicyPriority>
```

The collection of active policies that should be followed in the diabetes department is shown below.

```
<policy:Policy rdf:ID="DiabPolicy">
    <policy:actor rdf:resource="#var1"/>
```

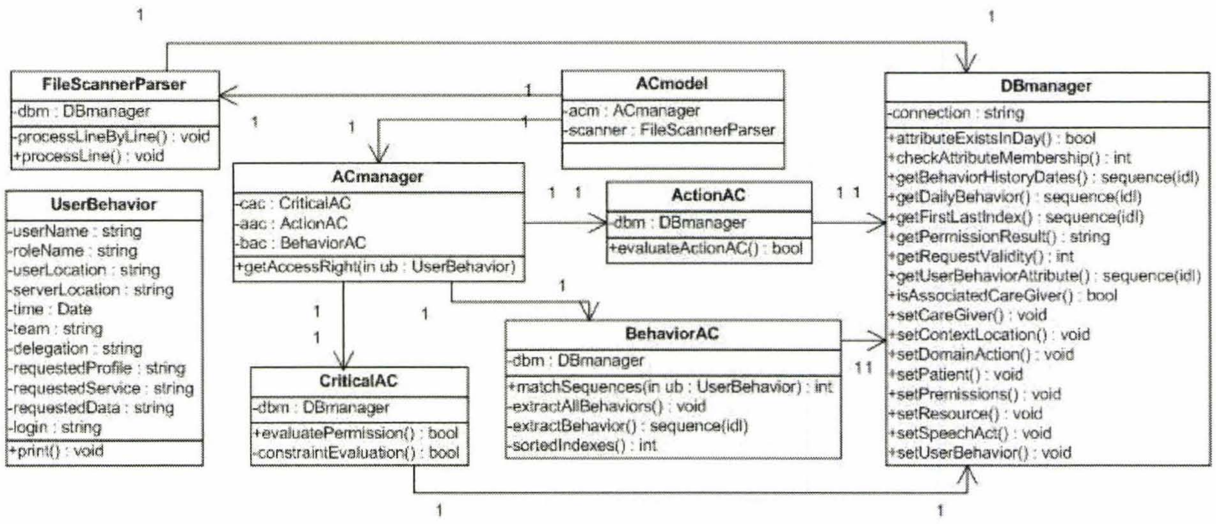

FileScannerParser
-dbm : DBmanager
-processLineByLine() : void
+processLine() : void
UserBehavior
-userName : string
-roleName : string
-userLocation : string
-serverLocation : string
-time : Date
-team : string
-delegation : string
-requestedProfile : string
-requestedService : string
-requestedData : string
-login : string
+print() : void
ACmodel
-acm : ACmanager
-scanner : FileScannerParser
ACmanager
-cac : CriticalAC
-aac : ActionAC
-bac : BehaviorAC
+getAccessRight(in ub : UserBehavior)
ActionAC
-dbm : DBmanager
+evaluateActionAC() : bool
CriticalAC
-dbm : DBmanager
+evaluatePermission() : bool
-constraintEvaluation() : bool
BehaviorAC
-dbm : DBmanager
+matchSequences(in ub : UserBehavior) : int
-extractAllBehaviors() : void
-extractBehavior() : sequence(idl)
-sortedIndexes() : int
DBmanager
-connection : string
+attributeExistsInDay() : bool
+checkAttributeMembership() : int
+getBehaviorHistoryDates() : sequence(idl)
+getDailyBehavior() : sequence(idl)
+getFirstLastIndex() : sequence(idl)
+getPermissionResult() : string
+getRequestValidity() : int
+getUserBehaviorAttribute() : sequence(idl)
+isAssociatedCareGiver() : bool
+setCareGiver() : void
+setContextLocation() : void
+setDomainAction() : void
+setPatient() : void
+setPremissions() : void
+setResource() : void
+setSpeechAct() : void
+setUserBehavior() : void
1
1

Figure 7.3: The class diagram of the classes of implementation

```
    <policy:context rdf:resource ="#IsMemberOfDiab"/>
    <policy:grants rdf:resource="#Perm_PhysDiabAction"/>
    <policy:grants rdf:resource="#Proh_PhysDiabAction"/>
    <policy:grants rdf:resource="#Perm_Jane"/>
    <policy:grants rdf:resource="#Perm_SmithDelegatePhysDiabAction"/>
    <policy:grants rdf:resource="#Granting_NursDiabAction"/>
    <policy:grants rdf:resource="#Obl_NurseSigningUp"/>
</policy:Policy>
```

7.3 Implementation classes

The classes that are used to implement the simulation environment are shown in the class diagram of Figure 7.3. This class diagram shows the interaction between the classes. In the following the functionality offered by each of these classes is discussed. Also the methods of the classes are explained.

ACmodel. This class initializes the model by entering the policy rules. The scanner and parser are called from the *FileScannerParser* class. Access requests are sent to the model through this class. These requests are objects of the *UserBehavior* class which are sent to the *ACmanager* class.

DBmanager. This class resides in the data layer and is the interface between the data layer and other classes. Its methods are as follows:

- Equivalent to each group of constraints in the *policy* file, there is a method that checks if input values occur in the constraint table (e.g. the *isAssociatedCareGiver* method).
- *setInstance* method is responsible for inserting the entities of the *instance* file into the database.
- *setPermission* method inserts different policies into the database. These policies belong to the domain action policies (introduced by Rei).
- *getRequestValidity* method checks if the input values exist in the database and returns a code based on the existence of the input values in the database.
- *getPermissionResult* method checks the *permission* table to see if a tuple exists in the table with the same values and input parameters (i.e. if the permission is defined for the user or not).
- *setSpeechAct* method is used to insert delegation permissions into the database.
- *setUserBehavior* method is used to insert user behavior values into the database.
- *setEntriesForBehavior* method is used to insert a simulated scenario into the database. This involves inserting new locations, teams, roles, resources, users, domain action permissions, and delegation permissions. The assignment between users-roles, users-teams, and roles-permissions should be included. These attributes are encoded in the action tuple format and are inserted into the database.

- *getUserBehaviorAttribute*, given a user and an attribute, returns all of the values that the attribute takes for the specified user.
- *getBehaviorHistoryDates*, given a user this method returns the list of days which are found in the user behavior history.
- *attributeExistsInDay*, given a user, an attribute, an attribute value, and a date, this method checks if the attribute takes that attribute value for the user during the specified day.
- *getFirstLastIndex*, given a user, an attribute, an attribute value, and a date, this method returns the *userBehaviorId* of the first and last occurrence of that attribute value for the attribute for the user in the specified day.
- *checkAttributeMembership*, given a *UserBehavior* object the validity of all of the attribute values is checked. The relation between the user and the team is also checked.
- *getDailyBehavior*, given a user, a date, and an attribute, this method returns the sequence of values that are recorded for that attribute from the beginning of the current day.

FileScannerParser. This class is responsible for inserting the policy rules from the Rei policy specification files into appropriate tables of the database. The class uses two methods to scan and parse the files. These methods parse the policy specification files and call the appropriate methods from the *DBmanager* class (*setInstance* and *setPermission* methods) to insert the values into the database. The parser looks for the entities of the *ontology* file in the *instances* and *policy* files (introduced in Section 7.2).

CriticalAC. This class performs the responsibilities that are defined for the *critical access control* block. Section 5.3.4 describes how to represent different policies. The major method of this class is *evaluatePermission*. This method first performs action authentication for the input request (i.e. checking the validity of input values) by calling the *getRequestValidity* method from the *DBmanager* class. If the input parameters are invalid, the appropriate error messages are returned. Otherwise, the *getPermissionResult* method from *DBmanager* is called to further process the input parameters for making the access control decision. This method is called with two different input values to check both specific and generic permissions. The final access decision result and the rule which authorized the user are returned as an output of the *evaluatePermission* method.

This method detects the inheritance relation between values of an attribute (e.g. if a given profile belongs to a certain category of profiles). Another method of this class is *constraintEvaluation* which examines if a given logical expression is true or not (e.g. if a user is a member of a certain department).

ActionAC. The only method of this class is *evaluateActionAC*. This method accepts an object of *UserBehavior* type as input. It then checks if all of the attributes of the given object have valid values. If the value provided for an attribute can not be detected as a valid value in the system (e.g. a team which does not exist in the database), access is denied. The user-team association is also checked here, i.e. if the user is not a member of the provided team, access is denied. The *checkAttributeMembership* method from *DBmanager* is used to deliver the above functionality.

BehaviorAC. This class performs the behavior analysis and access control decisions based on this analysis. It follows the algorithms introduced in Section 5.3.4. The *extract-Behavior* method extracts the behavior from a list of action tuples. This method accepts *userName, attributeName*, and *firstNvalues* as the input parameters. The output is the sequence of first *firstNvalues* attribute values for the attribute *attributeName* for the specified user. The algorithm for the *extractBehavior* method is shown in Algorithm 7. First, all of the values that the attribute takes for a given user are loaded. They are then sorted based on the frequency of their occurrence in different days and the first $firstNvalues$ values are selected.

A data structure (*attributePlaceHolder*) stores the order of appearance of these attributes in each day. Finally this data structure is reviewed to extract the appropriate attribute for each place to determine the final order. The other method of this class is *extractAllBehaviors*. This method calls the
extractBehavior method for all attributes of the action tuple for a certain user.

The *matchSequences* method accepts an object of *UserBehavior* type. The *extractBehavior* method is called to extract the common behavior of this user. The attribute values that each attribute has taken from the beginning of the current day are extracted. These two sequences (daily and common behavior) are compared, according to Algorithm 5, and a number is returned that represents the matching. The input object is stored in the database.

ACmanager. This class collects the results of other blocks of the *decision making engine* layer and produces the final decision. The *getAccessRight* method is responsible for performing this action. The *evaluateActionAC* method of the *ActionAC* class evaluates the validity of the attribute values. Then the *evaluatePermission* method of the *CriticalAC* class determines the availability of the requested permission according to stored policy rules. The *matchSequences* method of *BehaviorAC* then returns the matching of daily behavior and common behavior of the user. The *getAccessRight* method collects these results and returns the final access control decision of the model.

UserBehavior. This class represents the action tuple defined in Section 4.1. It contains the attributes of action tuple and provides setter and getter methods for them. The *print* method prints the values of all attributes.

7.4 Simulation result

In the following, the functionality of the classes described in the previous section are evaluated by test data. The test data is generated in a way that covers different cases (branches of workflow) of the targeted class.

7.4.1 Critical access control result

As discussed in Section 7.2, instances are introduced to the application in the *instance* file. The permissions are defined in the *policy* file. Suppose that we have the following instances and permissions:

Instances

DiabetesDep is an entity of *place* type.

Jane is a nurse who works in the Diabetes Department.

Julie is a nurse who works in the Diabetes Department.

Nero is a patient hospitalized in the Diabetes Department in room 223.

Jane is the nurse associated to *Nero*.

NeroEHR is Nero's electronic record.

DiabetesProfile represents the profile of patients hospitalized in the Diabetes Department.

Permissions

specific: *Jane* can perform the tasks allowed to associated nurses on *Nero*'s profile.

specific: *Jane* can perform common nurse actions on *Nero*'s profile.

generic: any nurse who is a member of the Diabetes Department can perform common nurse actions on instances of *DiabetesProfile*.

Having these instances and permissions, Jane has specific permission for different actions on Nero's profile. Julie has generic permission for performing common actions on Nero's profile. In order to evaluate the functionality of the application, different access request possibilities and their responses are given below.

Jane requests to perform a common nurse action on Nero's profile input: <Jane, ANursDiabAction, NeroEHR>

output: *ACCESS GRANTED with specific permission*

Jane requests to perform an action allowed to associated nurse, on Nero's profile

input: <Jane, ADiabAssociatedNursAction, NeroEHR>

output: *ACCESS GRANTED with specific permission*

Julie requests to perform a common nurse action on Nero's profile

input: <Julie, ANursDiabAction, NeroEHR>

output: *Not Authorized by specific permission*

output: *ACCESS GRANTED with generic inferred permission*

Julie requests to perform an action allowed to associated nurse, on Nero's profile

input: <Julie, ADiabAssociatedNursAction, NeroEHR>

output: *Not Authorized by specific permission*

output: *Not Authorized by generic permission either*

It can be observed that the application results match the outputs expected from the *critical access control* class for the given requests.

7.4.2 Behavior analysis

In order to test the functionality of the *BehaviorAC* class, we provide the class with simulated data and run the algorithms of the class over this data. The simulated data consists of the action tuples of a user. In this simulation, the activities of a nurse who works in the diabetes department is recorded for 5 shifts. The shifts occur once every two days between 8:00-17:00 and 00:00-8:00. Tables 7.3 and 7.4 show the entities used in this scenario. Figures 7.4 and 7.5 represent the action tuples for a day of the simulated scenario.

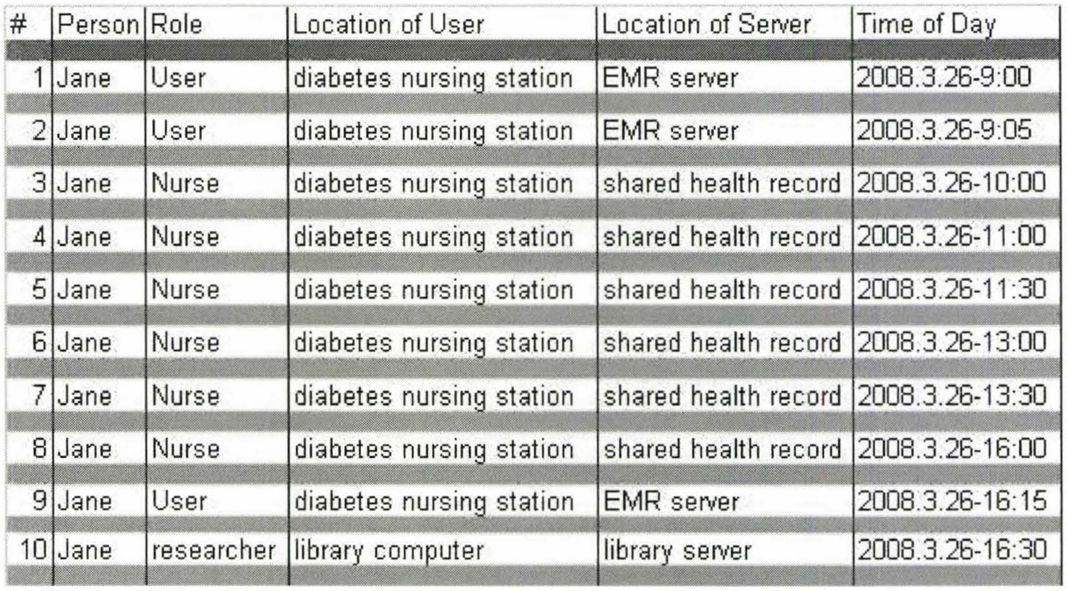
Here are the results of running the program over the simulated data (the results are

Simulation Entities (first part)			
Users	Roles	Location	Team
Jane	Nurse	diabeticDept	DiabeticNursingTeam
Julie	RegisteredUser	diabeticNursingStation	
Nadia	Emergency	EMRserver	
Nancy	Trainer	sharedHealthRecord	
Nero	Researcher	emergencyStation	
Natali		libraryServer	
Nemesis		libraryComputer	
Mike		laboratoryServer	
		activeRoleServer	

Table 7.3: Entities required for the behavior scenario - first part

Simulation Entities (second part)			
Delegation	Profiles	Services	Data types
Delg_DiaReview	DiabeticProfile	DiabReview	login
Delg_DiabDelegated	NadiaEHR	DiabDrugInjection	reminder
	NancyEHR	DiabGetAllergies	diabetic info
	NeroEHR	DiabSetAllergies	injection
	NataliEHR	DiabCheckUp	check up
	Nemesis	DiabTestResult	test results
	MikeEHR	DiabEmergencyCall	order
	DiabeticAccount	DiabDischarge	medical note
	JaneAccount	DiabSetUp	logout
	ActiveRoleDatabase	DiabLeaveNote	
	LibraryDatabase	DiabOrderLab	
	LaboratoryDatabase	SearchLibrary	
		GetAllergies	

Table 7.4: Entities required for the behavior scenario - second part



#	Person	Role	Location of User	Location of Server	Time of Day
1	Jane	User	diabetes nursing station	EMR server	2008.3.26-9:00
2	Jane	User	diabetes nursing station	EMR server	2008.3.26-9:05
3	Jane	Nurse	diabetes nursing station	shared health record	2008.3.26-10:00
4	Jane	Nurse	diabetes nursing station	shared health record	2008.3.26-11:00
5	Jane	Nurse	diabetes nursing station	shared health record	2008.3.26-11:30
6	Jane	Nurse	diabetes nursing station	shared health record	2008.3.26-13:00
7	Jane	Nurse	diabetes nursing station	shared health record	2008.3.26-13:30
8	Jane	Nurse	diabetes nursing station	shared health record	2008.3.26-16:00
9	Jane	User	diabetes nursing station	EMR server	2008.3.26-16:15
10	Jane	researcher	library computer	library server	2008.3.26-16:30

Figure 7.4: Sample data for a day of scenario - first part

modified from the actual application output, in a way that is easier to read). In the following the extracted sequence for each attribute is represented.

Call for roleId: 1: 5 (registeredUser), 2: 2 (nurse), 3: 8 (researcher)

Call for userLocation: 1: 6 (diabeticNursingStation), 2: 9 (emergencyStation), 3: 11 (libraryComputer)

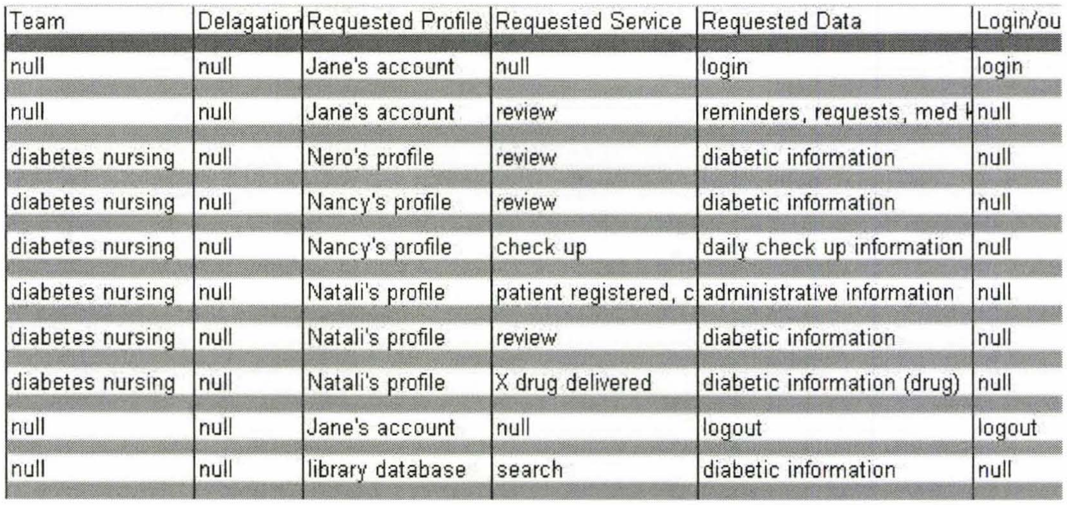
Call for serverLocation: 1: 7 (EMRserver), 2: 8 (sharedHealthRecord), 3: 12 (laboratoryServer), 4: 13 (activeRoleServer) 5: 10 (libraryServer)

Call for teamId: 1: 4 (DiabeticDeptNursingTeam)

Call for delegation: 1: 2 (Delg_DiabReview), 2: 6 (Delg_DiabDelegated)

Call for requestedProfile: 1: 15 (JaneAccount), 2: 8 (NadiaEHR), 3: 9 (NancyEHR), 4: 10 (NeroEHR), 5: 18 (LibraryDatabase)

Call for requestedService: 1: 18 (Review), 2: 19 (DiabReview), 3: 23 (DiabCheckUp), 4: 20 (DiabDrug-Injection), 5: 30 (SearchLibrary)



Team	Delagation	Requested Profile	Requested Service	Requested Data	Login/ou
null	null	Jane's account	null	login	login
null	null	Jane's account	review	reminders, requests, med k	null
diabetes nursing	null	Nero's profile	review	diabetic information	null
diabetes nursing	null	Nancy's profile	review	diabetic information	null
diabetes nursing	null	Nancy's profile	check up	daily check up information	null
diabetes nursing	null	Natali's profile	patient registered, c	administrative information	null
diabetes nursing	null	Natali's profile	review	diabetic information	null
diabetes nursing	null	Natali's profile	X drug delivered	diabetic information (drug)	null
null	null	Jane's account	null	logout	logout
null	null	library database	search	diabetic information	null

Figure 7.5: Sample data for a day of scenario - second part

Call for requestedData: 1: login, 2: reminders+requests+med knowledge, 3: diabetic information, 4: injection_diabetic information, 5: logout

Call for login: 1: login, 2: logout

Comparing the results with simulated input data, it can be observed that the algorithm successfully returns the attribute value sequence (common behavior) for given data.

7.4.3 Final result

In order to evaluate the model, the access requests that a user makes to the model during a day are simulated. The list of these requests is shown in Figure 7.6 (the figure shows a portion of the attributes). Examining these requests, the third request is not valid, as Jane is trying to access the record of a patient who is not hospitalized in the Diabetes Department. The fifth request is also not valid, as Jane is asking for a type of delegation (that is only permitted to physicians) which she is not allowed to perform. The ninth request is invalid, since Jane is not registered as a member of the diabetes nursing team. Other requests follow the access control rules and will gain access to their requested resource.

In the following, the access requests of Figure 7.6 are submitted to the model sequentially. For each request, a portion of the results of the blocks of the *decision making engine* layer is shown. The *action access control* block checks the validity of arguments. If the value provided for an attribute can not be detected as a valid value in the system, access is denied to maintain data integrity. The *critical access control* block evaluates the available permissions according to privacy and security rules. *Behavior access control* extracts the common expected behavior of the user and also the daily behavior. It then compares these two behaviors and returns a number representing their match (here we just consider the *role* attribute). *Access control manager* accepts an action tuple as input and invokes appropriate methods of the classes mentioned in this paragraph, to make the final decision. The input action tuple and the results of each of the above blocks are printed.

time:09:00:00.0, user:Jane, role:registeredUser, user location:diabeticNursingStation, server location: EMRserver, requested profile:JaneAccount, login:login

Critical access control: Login/out event occurred!

Action access control: All arguments are valid!

Behavior access control: Common sequence: 5-2-8, Daily behavior: 5, Return value: 1

time:09:05:00.0, user:Jane, role:registeredUser, user location:diabeticNursingStation, server location: EMRserver, requested profile:JaneAccount, requested service:Review, requested data: reminders+med knowledge

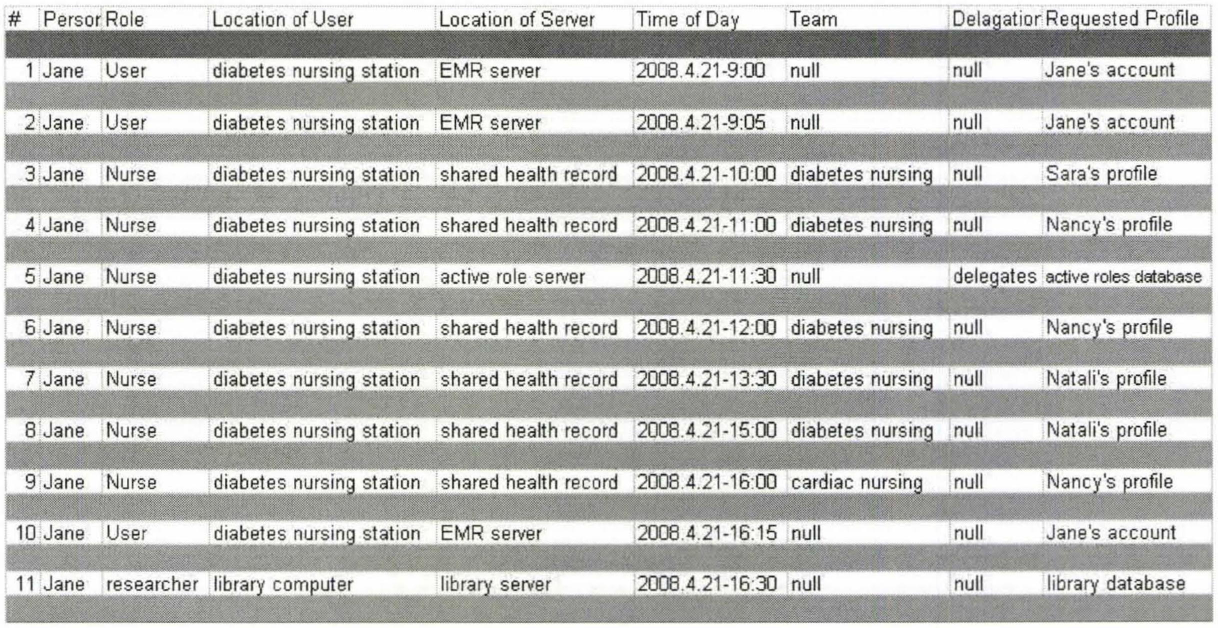
Critical access control:

Not Authorized by specific permission

Not Authorized by generic permission either

Action access control: All arguments are valid!

Behavior access control: Common sequence: 5-2-8, Daily behavior: 5, Return value: 1



#	Persor	Role	Location of User	Location of Server	Time of Day	Team	Delagatior	Requested Profile
1	Jane	User	diabetes nursing station	EMR server	2008.4.21-9:00	null	null	Jane's account
2	Jane	User	diabetes nursing station	EMR server	2008.4.21-9:05	null	null	Jane's account
3	Jane	Nurse	diabetes nursing station	shared health record	2008.4.21-10:00	diabetes nursing	null	Sara's profile
4	Jane	Nurse	diabetes nursing station	shared health record	2008.4.21-11:00	diabetes nursing	null	Nancy's profile
5	Jane	Nurse	diabetes nursing station	active role server	2008.4.21-11:30	null	delegates	active roles database
6	Jane	Nurse	diabetes nursing station	shared health record	2008.4.21-12:00	diabetes nursing	null	Nancy's profile
7	Jane	Nurse	diabetes nursing station	shared health record	2008.4.21-13:30	diabetes nursing	null	Natali's profile
8	Jane	Nurse	diabetes nursing station	shared health record	2008.4.21-15:00	diabetes nursing	null	Natali's profile
9	Jane	Nurse	diabetes nursing station	shared health record	2008.4.21-16:00	cardiac nursing	null	Nancy's profile
10	Jane	User	diabetes nursing station	EMR server	2008.4.21-16:15	null	null	Jane's account
11	Jane	researcher	library computer	library server	2008.4.21-16:30	null	null	library database

Figure 7.6: Sample requests made to the model

time:10:00:00.0, user:Jane, role:Nurse, user location:diabeticNursingStation, server location: sharedHealthRecord, team:DiabeticDeptNursingTeam, requested profile: SaraEHR, requested service: DiabReview, requested data:cardiac information

Critical access control:

Not Authorized by specific permission

Not Authorized by generic permission either

Action access control: All arguments are valid!

Behavior access control: Common sequence: 5-2-8, Daily behavior: 5-2, Return value: 1

time:11:00:00.0, user:Jane, role:Nurse, user location:diabeticNursingStation, server location: sharedHealthRecord, team:DiabeticDeptNursingTeam, requested profile: NancyEHR, requested service: DiabReview, requested data:diabetic information

Critical access control: ACCESS GRANTED with generic inferred permission

Action access control: All arguments are valid!

Behavior access control: Common sequence: 5-2-8, Daily behavior: 5-2, Return value: 1

time:11:30:00.0, user:Jane, role:Nurse, user location:diabeticNursingStation, server location:

activeRoleServer, team:DiabeticDeptNursingTeam, delegation: DelgDiaPhys, requested profile: activeRoleDatabase

Critical access control: Invalid request: Unknown operation-Invalid service value

Action access control: Invalid arguments!

time:12:00:00.0, user:Jane, role:Nurse, user location:diabeticNursingStation, server location: sharedHealthRecord, team:DiabeticDeptNursingTeam, requested profile: NancyEHR, requested service: DiabCheckUp, requested data:checkUp diabetic information

Critical access control: ACCESS GRANTED with generic inferred permission

Action access control: All arguments are valid!

Behavior access control: Common sequence: 5-2-8, Daily behavior: 5-2, Return value: 1

time:13:30:00.0, user:Jane, role:Nurse, user location:diabeticNursingStation, server location: sharedHealthRecord, team:DiabeticDeptNursingTeam, requested profile: NataliEHR, requested service: DiabReview, requested data:diabetic information

Critical access control: ACCESS GRANTED with generic inferred permission

Action access control: All arguments are valid!

Behavior access control: Common sequence: 5-2-8, Daily behavior: 5-2, Return value: 1

time:15:00:00.0, user:Jane, role:Nurse, user location:diabeticNursingStation, server location: sharedHealthRecord, team:DiabeticDeptNursingTeam, requested profile: NataliEHR, requested service: DiabDrugInjection, requested data:injection diabetic information

Critical access control: ACCESS GRANTED with generic inferred permission

Action access control: All arguments are valid!

Behavior access control: Common sequence: 5-2-8, Daily behavior: 5-2, Return value: 1

time:2008-05-21 16:00:00.0, user:Jane, role:Nurse, user location:diabeticNursingStation, server location:sharedHealthRecord, team:CardiacDeptNursingTeam, requested profile: NancyEHR, requested service:DiabCheckUp, requested data:checkUp_diabetic information

Action access control: Invalid arguments! Invalid team value

time:16:15:00.0, user:Jane, role:registeredUser, user location:diabeticNursingStation, server location: EMRserver, requested profile:JaneAccount, login:logout

Critical access control: Login/out event occurred!

Action access control: All arguments are valid!

Behavior access control: Common sequence: 5-2-8, Daily behavior: 5-2, Return value: 1

time:16:30:00.0, user:Jane, role:Researcher, user location:libraryComputer, server location:libraryServer, requested profile:LibraryDatabase, requested service:SearchLibrary, requested data:diabetic new information

Critical access control: ACCESS GRANTED with generic inferred permission

Action access control: All arguments are valid!

Behavior access control: Common sequence: 5-2-8, Daily behavior: 5-2-8, Return value: 1

Chapter 8

Conclusion

In this thesis we presented a framework and detailed steps for provision of the security aspects in distributed healthcare systems. The model is generic in the sense that it has a layer that allows the user to map environment specific contexts onto a standard internal set of contexts. Consequently, these contexts are fed into an access control engine to evaluate the authorization merits of the corresponding user. The proposed access control method models the user's action as a tuple of major contexts which in turn allows us to demonstrate different attributes of the user such as: *single action*, *daily behavior*, and *snapshot behavior*. These cover three dimensions of the user's activities that can also be viewed as: static, dynamic and historical aspects of the user's activities. The architecture proposed for the model consists of several layers, each responsible for a different access control functionality.

The model is designed to be compatible with the international healthcare information model HL7 RIM. It extends RIM's class diagram to incorporate the proposed behavior-based access control. The proposed model satisfies requirements of the healthcare domain such as patient consent, authorization in emergency situations, auditing of all events and considering care givers activities. It also uses the recent results of healthcare standards (documents for roles, contexts and scenarios) and is compliant with their technical specifications. The dynamic and flexible nature of the model helps to deal with the inherent heterogeneity of distributed healthcare systems.

In terms of future work, there are several important avenues:

- The available technologies and equipment should be reviewed to extract a minimum set of technologies that allows our model to extract the required contexts.
- The proposed model should be applied to a real world case study to evaluate the functionality of the model in the following aspects:
  - The interoperability of the model should be evaluated by testing the proposed data model of security effective factors. The security factors of two organizations should be mapped to this data model and inter-organizational requests should be generated.
  - The extracted *common behavior* and the real behavior of users should be compared. A method should be designed to match these two behaviors in order to be able to evaluate the model.
  - The correctness of suggestions and warnings generated by the model should be examined by analyzing the data stored in the *audit trail* block. In case of detecting an irrelevant warning, the algorithms introduced in this thesis should be revisited to be improved.
- The analysis and algorithms introduced for the components of the architecture can be improved to determine more complex behaviors and better use them to make access control decisions. This might require extending the definition of *user behavior*.
- As another application domain, the *user behavior* concept and its corresponding model can be employed to provide guidelines for care givers based on their behavior.

In this way, a user is able to gain recommendations based on his behavior and his colleagues' behaviors.

Appendix A

Algorithms

Algorithm 1 Role Checking (user,currentPermission)

```
1: $session \leftarrow SessionUser^{-1}(user)$
2: $RoleArray \leftarrow SessionRole(session)$
3: for all $r \in RoleArray$ do
4:     $PermissionArray \leftarrow$ all permissions that are in relation $PA$ with $r$
5:     if $currentPermission \in PermissionArray$ then
6:         return true
7:     end if
8: end for
9: return false
```

Algorithm 2 Team Checking (user, currentTeam, currentRole, currentPermission)

```
1: if $< currentRole, currentTeam > \in RT$ then
2:   if $< user, < currentRole, currentTeam >> \in UT$ then
3:     $PermissionArray \leftarrow$ all permissions that are in relation $PA$ with $currentRole$
4:     if $currentPermission \in PermissionArray$ then
5:       return true
6:     end if
7:   end if
8: end if
9: return false
```

Algorithm 3 Action Access Control (user, currentAction)

```
1: $actionMembershipArray \leftarrow 0$
2: for every $att \in$ action tuple attributes do
3:     $attFound \leftarrow$ false
4:     for every $action \in B(user)$ do
5:         if (value of $att$ in $action$ = value of $att$ in $currentAction$) and (not $attFound$)
           then
6:             append 1 to $actionMembershipArray$
7:             $attFound \leftarrow$ true
8:         end if
9:     end for
10:    if not $attFound$ then
11:        append 0 to $actionMembershipArray$
12:    end if
13: end for
14: return $actionMembershipArray$
```

Algorithm 4 Extract behavior sequence for an attribute value from a set of action tuples

```
1: List the attribute values that appear in user action records for given attribute
2: for each day do
3:     Initialize an instance of the list (of step 1) to 0
4:     Mark an attribute value as 1 if the user has used it in this day
5: end for
6: Sum up the list of different days (used in steps 2-5) to get an accumulative list for
   attribute values
7: Find the attribute values with the greatest frequency of occurrence (first $N$, where
   $N$ is the length of the behavior sequence)
8: Compare the sequences of the first $N$ attribute values in these days and pick the
   sequence occurring more frequently (common sequence)
```

Algorithm 5 Sequence matching

```
1: $correctOrders \leftarrow 0$
2: $n \leftarrow$ size of common behavior sequence
3: for $i \leftarrow 1$ to $n$ do
4:     if (order of commonBehavior[i] and commonBehavior[i+1] is the same in
       dailyBehavior) then
5:         $correctOrders \leftarrow correctOrders + 1$
6:     end if
7: end for
8: return $correctOrders/n$
```

Algorithm 6 Access Control Manager

```
1: $criticalCheck \leftarrow$ true
2: if (not authorized by role checking and context checking and not authorized by team membership and not authorized by delegation) then
3:     $criticalCheck \leftarrow$ false
4: end if
5: $behaviorCredit \leftarrow loadBehaviorCredit()$, $criticalCredit \leftarrow loadCriticalCredit()$
6: if behavior check result > threshold then
7:     $behaviorCredit \leftarrow behaviorCredit + 1$
8:     if criticalCheck then
9:         $criticalCredit \leftarrow criticalCredit + 1$
10:        return access granted + behavior result
11:    else
12:        $criticalCredit \leftarrow criticalCredit - 1$
13:        Suspicious behavior detected + $criticalCredit$ + $behaviorCredit$
14:        return access denied
15:    end if
16: else
17:    $behaviorCredit \leftarrow behaviorCredit - 1$
18:    if criticalCheck then
19:        $criticalCredit \leftarrow criticalCredit - 1$
20:        Notify the security administrator + $criticalCredit$ + $behaviorCredit$
21:        return access granted + behavior result
22:    else
23:        $criticalCredit \leftarrow criticalCredit + 1$
24:        return access denied
25:    end if
26: end if
```

Algorithm 7 Behavior extraction

```
$attributeValues \leftarrow getUserBehaviorAttribute()$
$historyDays \leftarrow getBehaviorHistoryDates()$
$attributeOccurrence \leftarrow 0$
for $i \leftarrow 1$ to length[$historyDays$] do
    for $j \leftarrow 1$ to length[$attributeValues$] do
        if $attributeExistsInDay()$ then
            $attributeOccurrence[j] \leftarrow attributeOccurrence[j] + 1$
        end if
    end for
end for
$sortedAttributeIndexes \leftarrow$ index of sorted elements of $attributeOccurrence$
$attributePlaceHolder \leftarrow$ the data structure storing places of attribute values in a day initialized to 0
for $i \leftarrow 1$ to length[$historyDays$] do
    for $j \leftarrow 1$ to firstNvalues do
        $attributeSequence \leftarrow getFirstLastIndex$ of $sortedAttributeIndex[j]$ for day $i$
    end for
    $dailySortedArrayListIndexes \leftarrow$ index of sorted $attributeSequence$ for day $i$
    for $k \leftarrow 1$ to firstNvalues do
        $attributeValuePlaceHolder[dailySortedArrayListIndexes[k]][\text{firstNvalues} - k - 1] \leftarrow attributeValuePlaceHolder[dailySortedArrayListIndexes[k]][\text{firstNvalues} - k - 1] + 1$
    end for
end for
for $i \leftarrow 1$ to firstNvalues do
    print attribute value with greatest $attributePlaceHolder[i]$ not printed before
end for
```

Glossary

ABAC-Attribute Based Access Control	ABAC is an access control model that considers user's attributes for deciding about access control, 23
AOP-Aspect Oriented Programming	AOP is based on the idea that computer systems are better programmed by separately specifying the various concerns (properties or areas of interest) of a system and some description of their relationships, and then relying on mechanisms in the underlying AOP environment to weave or compose them together into a coherent program, 25
CAAC-Context Aware Access Control	CAAC is an access control model that considers user's context for deciding about access control, 25
CAS-Context-aware Systems	A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task, 27
CBAC-Content Based Access Control	CBAC is an access control model that considers resource's content for deciding about access control, 23
CSAC-Context Sensitive Access Control	CSAC is an access control model that considers user's context for authentication and authorization, 25
D-MIM - Domain Message Information Model	D-MIM is a subset of the RIM that includes a fully expanded set of class clones, attributes and relationships that are used to create messages for any particular domain, 9

EHR-Electronic Health Record	An EHR provides each individual patient with a secure and private lifelong record of their health history, 12
EHRi-Electronic Health Record Infostructure	EHRi is a collection of common and reusable components in the support of a diverse set of health information management applications, 12
EHRS-Electronic Health Record Solution	EHRS is a combination of people, organizational entities, business processes, systems, technologies and standards that interact and exchange clinical data to provide effective healthcare, 12
EMR-Electronic Medical Record	User's medical record in digital format, 67
HIAL-Health Information Access Layer	HIAL provides a single standardized way for Point of Service applications to connect to the EHRi, regardless of how a particular jurisdiction has partitioned EHR information domains and services, 12
HIPAA	Health Insurance Portability and Accountability Act is a healthcare standard that provides privacy requirements, 16
HL7 RIM-Reference Information Model	An object model that is a representation of clinical data and identifies the life cycle of the events that a message will carry, 8
HL7-Health Level Seven	An international community of healthcare experts and information scientists collaborating to create standards for the exchange, management and integration of electronic healthcare information, 8
HMD-Hierarchial Message Description	HMD is a tabular representation of the sequence of elements represented in an R-MIM. Each HMD produces a single base message template from which the specific message types are drawn, 9

IHE-Integrating the Healthcare Enterprise	IHE is an initiative designed to promote standard-based methods of data integration in healthcare and encompasses industry and users, 17
Infoway-Canada Health Infoway	Infoway is an organization that provides specifications for a standard, nationwide healthcare infostructure, 11
LOINC	Logical Observation Identifiers, Names and Codes is a clinical terminology system, 17
PHI-Personal Health Information	Health information of individuals, 14
Ponder	Ponder is a policy specification language, 81
PSA-Privacy and Security Architecture	An Infoway group that is responsible for developing the security standards and maintaining information privacy, 13
R-MIM - Refined Message Information Model	R-MIM is a subset of the D-MIM and contains only those classes, attributes and associations required to compose the set of messages, 9
RBAC-Role Based Access Control	RBAC is an access control model that considers user's role for deciding about access control, 22
Rei	Rei is a policy specification language., 82
SAAC-Situation Aware Access Control	SAAC is an access control model that considers user's situation for deciding about access control, 23
SBAC-Scenario Based Access Control	SBAC is an access control model that considers scenarios for deciding about access control, 24
SNOMED	Systematized Nomenclature of Medicine is a comprehensive clinical terminology system that provides clinical content and expressivity for clinical documentation and reporting, 16
SOA-Service Oriented Architecture	A computer system architecture that is composed of services, 1

SoD-Separation of Duty	Separation of Duty is the concept of having more than one person required to complete a task. More specifically it is defined as disseminating the tasks and associated privileges for a specific business process among multiple users, 50
SoS-System of Systems	SoS are large-scale concurrent and distributed systems that are comprised of complex systems. SoS integration is a method to pursue in development, integration, interoperability, and optimization of systems to enhance performance in future scenarios, 3
TBAC-Team Based Access Control	TBAC is an access control model that considers user's team for deciding about access control, 22

Bibliography

[1] Canada Health Infoway official website. URL = www.infoway-inforoute.ca. [Online; accessed 1-April-2008].

[2] Canadian Nurses Associatino (CNA) website. URL = www.cna-aiic.ca. [Online; accessed 1-April-2008].

[3] Health Insurance Portability and Accountability Act (HIPAA). URL = www.hipaa.org. [Online; accessed 1-April-2008].

[4] Health Level Seven Ballot official website. URL = www.hl7.org/v3ballot/html/welcome/environment/index.htm. [Online; accessed 1-April-2008].

[5] Health Level Seven official website. URL = www.hl7.org. [Online; accessed 1-April-2008].

[6] Integrating the Healthcare Enterprise (IHE) official website. URL = www.ihe.net. [Online; accessed 1-April-2008].

[7] Logical Observation Identifiers Names and Codes (LOINC) official website. URL = www.regenstrief.org/medinformatics/loinc. [Online; accessed 1-April-2008].

[8] Open Clinical website. URL = www.openclinical.org. [Online; accessed 1-April-2008].

[9] Ponder toolkit website. URL = http://ponder2.net/. [Online; accessed 1-April-2008].

[10] Protege website. URL = http://protege.stanford.edu/. [Online; accessed 1-April-2008].

[11] Rei website. URL = http://rei.umbc.edu/. [Online; accessed 1-April-2008].

[12] TouchGraph website. URL = www.touchgraph.com. [Online; accessed 1-April-2008].

[13] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic databases. In *VLDB '02: 28th International Conference on Very Large Databases*, pages 143–154, 2002.

[14] Jalal Al-Muhtadi, Anand Ranganathan, Roy Campbell, and M. Dennis Mickunas. Cerberus: A context-aware security scheme for smart spaces. In *PERCOM '03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, page 489, 2003.

[15] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing, forthcoming*, 2(4):263–277, 2007.

[16] Jakob E. Bardram. Applications of context-aware computing in hospital work: examples and design principles. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 1574–1579, 2004.

[17] Rafae Bhatti, Elisa Bertino, and Arif Ghafoor. A trust-based context-aware access control model for web-services. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services*, page 184, 2004.

[18] Rafae Bhatti, Arif Ghafoor, Elisa Bertino, and James B. D. Joshi. X-gtrbac: an xml-based policy specification framework and architecture for enterprise-wide ac-

cess control. *ACM Transactions on Information and System Security (TISSEC)*, 8(2):187–227, 2005.

[19] Rafae Bhatti, Khalid Moidu, and Arif Ghafoor. Policy-based security management for federated healthcare databases (or RHIOs). In *HIKM '06: Proceedings of the international workshop on Healthcare information and knowledge management*, pages 41–48, 2006.

[20] Bernd Blobel. Trustworthiness in distributed Electronic Healthcare Records - basis of shared care. In *Computer Security Applications Conference*, pages 433–441, 2001.

[21] Bernd Blobel. Authorization and access control for electronic health record systems. *International Journal of Medical Informatics*, 73:251–257, 2004.

[22] Reinhardt A. Botha and Jan H.P. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666–682, 2001.

[23] CNA. Advances nursing practice - a national framework, April 2002.

[24] LOINC Committee. LOINC users' guide, June 2007.

[25] Ernesto Damiani, Sabrina De Capitani di Vimercati, and Pierangela Samarati. New paradigms for access control in open environments. In *Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology*, pages 540–545, 2005.

[26] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, pages 18–38, 2001.

[27] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Computing*, 5(1):4–7, 2001.

[28] Tzilla Elrad, Robert E. Filman, and Atef Bader. Aspect-oriented programming: Introduction. *Communications of the ACM*, 44(10):29–32, 2001.

[29] David F. Ferraiolo, D. Richard Kuhn, and Ramaswamy Chandramouli. *Role-Based Access Control*. Artech House, 2003.

[30] Ana Ferreira, Ricardo Cruz-Correia, Luis Antunes, and David Chadwick. Access control: How can it improve patients' healthcare? *Study in Health Technology and Informatics*, 127:65–76, 2007.

[31] Ernest Friedman-Hill. Jess, the rule engine for java platform. Sandia Natinoal Laboratories, 2005.

[32] Dragan Gasevic, Dragan Djuric, Vladan Devedzic, and Violeta Damjanovi. Converting UML to OWL ontologies. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 488–489, 2004.

[33] Christos K. Georgiadis, Ioannis Mavridis, George Pangalos, and Roshan K. Thomas. Flexible team-based access control using contexts. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 21–27, 2001.

[34] Luigi Giuri and Pietro Iglio. Role templates for content-based access control. In *RBAC '97: Proceedings of the second ACM workshop on Role-based access control*, pages 153–159, 1997.

[35] HIPAA. Security standards: Technical safeguards, March 2007. version 2.

[36] HL7. Message development framework, December 1999. version 3.3.

[37] HL7. RBAC healthcare scenarios, November 2005. v2.

[38] HL7. RBAC role engineering process, November 2005. v1.1.

[39] HL7. HL7 healthcare scenario roadmap, September 2006. v2.2.

[40] Mario Hoffmann, Atta Badii, Stephen Engberg, Renjith Nair, Daniel Thiemert, and Manuel Matthess. Security, trust and privacy supported by context-aware middleware. 2007.

[41] Junzhe Hu and Alfred C. Weaver. A dynamic, context-aware security infrastructure for distributed healthcare applications. In *Proceedings of the First Workshop on Pervasive Privacy Security, Privacy, and Trust*, 2004.

[42] R. J. Hulsebosch, A. H. Salden, M. S. Bargh, P. W. G. Ebben, and J. Reitsma. Context sensitive access control. In *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 111–119, 2005.

[43] Patrick Hung. Towards a privacy access control model for e-healthcare services. In *PST '05: Third Annual Conference on Privacy, Security and Trust*, 2005.

[44] Patrick C. K. Hung, Jude Andrade, Yongming Chen, Ranny Huang, Miguel Vargas Martin, and Yi Zheng. Research issues of privacy access control model for mobile ad hoc healthcare applications with xacml. In *AINA '07: Advanced Information Networking and Applications Workshops*, pages 582–587, 2007.

[45] Canada Health Infoway. EHR Privacy and Security Requirements, February 2005. v1.1.

[46] Canada Health Infoway. EHRi Privacy and Security Conceptual Architecture, June 2005. v2.

[47] Canada Health Infoway. EHRS Blueprint, an interoperable EHR framework, April 2006. v2.

[48] Canada Health Infoway. iEHR Scope and Package Tracking Framework, April 2007. IE50102-PM99.

[49] Canada Health Infoway. IHE IT infrastructure techinical framework - volume 1 - integration profile, August 2007. revision 4.

[50] Jens H. Jahnke. Toward context-aware computing in clinical care. In *OOPSLA Workshop on Building Software for Pervasive Computing*, 2005.

[51] Jens H. Jahnke, Yury Bychkov, David Dahlem, and Luay Kawasme. Context-aware information services for health care. In *Revue d'Intelligence Artificielle*, volume 19, pages 459–478, 2005.

[52] Myong H. Kang, Joon S. Park, and Judith N. Froscher. Access control mechanisms for inter-organizational workflow. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 66–74, 2001.

[53] Kapsalis, V. Karelis, D. Hadellis, and L. Papadopoulos. A context-aware access control framework for e-service provision. In *Industrial Technology, 2005. ICIT 2005. IEEE International Conference on*, pages 932–937, 2005.

[54] C. Joncheng Kuo and Polar Humenn. Dynamically authorized role-based access control for secure distributed computation. In *XMLSEC '02: Proceedings of ACM workshop on XML security*, pages 97–103, 2002.

[55] Xueli Li, Nomair A. Naeem, and Bettina Kemme. Fine-granularity access control in 3-tier laboratory information systems. In *IDEAS '05: Proceedings of the 9th International Database Engineering & Application Symposium*, pages 391–397, 2005.

[56] Gustaf Neumann and Mark Strembeck. A scenario-driven role engineering process for functional RBAC roles. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 33–42, 2002.

[57] U. Nitsche, R. Holbein, O. Morger, and S. Teufel. Realization of a context-dependent access control mechanism on a commercial platform. In *IFIP/Sec'98: Proceedings of the Fourtheenth International Information Security Conference*, pages 160–170, 1998.

[58] College of American Pathologists. SNOMED clinical terms guide - abstract logical models and representational forms, January 2006. version 5.

[59] College of American Pathologists. SNOMED clinical terms user guide, January 2007.

[60] Wan rong Jih, Shao-You Cheng, Jane Yung jen Hsu, and Tse-Ming Tsai. Context-aware access control on pervasive healthcare. In *MAM '05: Mobility, Agents, and Mobile Services*, pages 21–28, 2005.

[61] Lillian Rostad and Ole Edsberg. A study of access control requirements for healthcare systems based on audit trails form access logs.

[62] Ferat Sahin, Mo Jamshidi, and Prassana Sridhar. A discrete event xml based simulation framework for system of systems. In *SoSE '07: IEEE International Conference on System of Systems Engineering*, pages 1–7, 2007.

[63] Arjmand Samuel. Context-aware access control policy engineering for electronic health records. In *Research Seminar at CIMIC*, 2007.

[64] Dirk Schwartmann. An attributable role-based access control for healthcare. In *ICCS '04: International Conference on Computational Science*, pages 1148–1155, 2004.

[65] Claudia Linnhoff-Popien Thomas Strang. A context modeling survey. In *Workshop Proceedings, First International Workshop on Advanced Context Modelling*, 2004.

[66] Alessandra Toninelli, Rebecca Montanari, Lalana Kagal, and Ora Lassila. A semantic context-aware access control framework for secure collaborations in pervasive

computing environments. In *ISWC '06: Fifth International Semantic Web Conference*, pages 473–486, 2006.

[67] Marc Wilikens, Simone Feriti, Alberto Sanna, and Marcelo Masera. A context-related authorization and access control method based on RBAC. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 117–124, 2002.

[68] Thida Win, Hninn Mar Aung, and Ni Lar Thein. An MDA based approach for facilitating representation of semantic web service technology. In *APSITT 05: Proceeding of Information and Telecommunication Technologies*, pages 260–265, 2005.

[69] Konrad Wrona and Laurent Gomez. Context-aware security and secure context-awareness in ubiquitous computing environments. In *Autumn Meeting of Polish Information Processing Society Conference Proceedings*, pages 255–265, 2005.

[70] Stephen S. Yau, Yisheng Yao, and Vageesh Banga. Situation-aware access control for service-oriented autonomous decentralized systems. In *ISADS '05: Proceedings of Autonomous Decentralized Systems*, pages 17–24, 2005.

[71] Guangsen Zhang and Manish Parashar. Dynamic context-aware access control for grid applications. In *GRID '03: Proceedings of the Fourth International Workshop on Grid Computing*, page 101, 2003.

[72] Longhua Zhang, Gail-Joon Ahn, and Bei-Tseng Chu. A role-based delegation framework for healthcare information systems. In *SACMAT '02: Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, pages 125–134, 2002.